Practical Large-Scale Proof-of-Stake Asynchronous Total-Order Broadcast

Orestis Alpos
University of Bern
orestis.alpos@unibe.ch

Christian Cachin
University of Bern
christian.cachin@unibe.ch

Simon Holmgaard Kamp
Aarhus University
kamp@cs.au.dk

Jesper Buus Nielsen
Aarhus University
jbn@cs.au.dk

Abstract

We present simple and practical protocols for generating randomness as used by asynchronous total-order broadcast. The protocols are secure in a proof-of-stake setting with *dynamically changing* stake. They can be plugged into existing protocols for asynchronous total-order broadcast and will turn these into asynchronous total-order broadcast with dynamic stake. Our contribution relies on two important techniques. The paper "Random Oracles in Constantinople: Practical Asynchronous Byzantine Agreement using Cryptography" [Cachin, Kursawe, and Shoup, PODC 2000] has influenced the design of practical total-order broadcast through its use of threshold cryptography. However, it needs a setup protocol to be efficient. In a proof-of-stake setting with dynamic stake this setup would have to be continually recomputed, making the protocol impractical. The work "Asynchronous Byzantine Agreement with Subquadratic Communication" [Blum, Katz, Liu-Zhang, and Loss, TCC 2020] showed how to use an initial setup for broadcast to asymptotically efficiently generate sub-sequent setups. The protocol, however, resorted to fully homomorphic encryption and was therefore not practically efficient. We adopt their approach to the proof-of-stake setting with dynamic stake, apply it to the Constantinople paper, and remove the need for fully homomorphic encryption. This results in simple and practical proof-of-stake protocols. We discuss how to use the new coin-flip protocols together with *DAG rider* [Keidar et al., PODC 2021] and create a variant which works for dynamic proof of stake. Our method can be employed together with many further asynchronous total-order broadcast protocols.

1 Introduction

State of the art. It is well known that Asynchronous Total-Order Broadcast (ATOB) cannot be deterministic [25]. The necessary randomness is usually modelled as a *common coin* scheme [34], informally defined as a source random values observable by all participants but unpredictable for the adversary [9]. Common coins are most practically implemented using threshold cryptography [10, 23, 33, 9]. This approach has many benefits. It is conceptually simple and efficient, it achieves optimal resilience $t < n/3$, where n the number of parties running the protocol, and it results in a *perfect coin*, meaning that it is uniformly distributed and agreed-upon with probability 1. The drawback, however, is that it requires a trusted setup or an Asynchronous Distributed

Key Generation (ADKG) protocol. Current state of the art ADKG protocols [20, 1, 2] have communication cost of $O(\lambda n^3)$, where λ is the security parameter.

Given that state-of-the-art ATOB protocols have communication complexity $O(\lambda n^2)$, or even amortized $O(\lambda n)$, it is evident that the communication cost of ADKG becomes the bottleneck. In a permissioned setting with a static set of parties, it is common to proactively refresh the threshold setup [8]. In a Proof-of-Stake (PoS) setting, particularly, where the stake is constantly evolving and parties may dynamically join or leave the protocol, the ADKG protocol must be run periodically. Recent literature on asynchronous consensus uses committees, which contain only a subset of the parties, reducing the communication complexity of BA even further to $O(\lambda n \log n)$ at the cost of tolerating only $t < (1 - \epsilon)n/3$ corruptions for any $\epsilon > 0$ [4, 17]. As the protocol run by the committee assumes an honest supermajority, this paradigm comes with one of two significant drawbacks. Either the sampled committee has to be very large, so that its maximal corruption remains below $n/3$ with overwhelming probability [22]. Otherwise, in order to keep the committee size small, the corruption level in the ground population must be assumed lower than $n/3$ by a considerable margin. Directly porting this idea to ADKG results in the same drawbacks. Finally, existing DKG protocols support only flat structures, where every party has the same weight and in total $t < n/3$ parties are corrupted. They do not readily work for a setting where every party holds a different share of the stake.

Seeds in PoS protocols. PoS-based ATOB protocols and blockchains require, apart from common coins, a second type of randomness, usually referred to as a *seed.* In PoS blockchains there is the notion of *accounts* with stake on them, of *roles*, such as "produce the 42-nd block", and of a *lottery*, through which accounts win the right to execute roles. This is typically [21, 26] implemented using a *Verifiable Pseudo-Random Function* (VRF) [31]: each account has a private key for a VRF and applies it to the role, producing a pseudorandom value. If this value is above a threshold then the account wins the right to execute the role. However, for this approach to work the lottery needs as input not only a role but also a seed. Without the seed, a party can operate with several accounts and move all its stake to the luckiest account. By including a seed in the lottery and using the stake distribution from a point in time *before* the seed was unpredictable one can mitigate this attack [21].

In practice one can use a common-coin protocol to produce the seeds. We remark, however, that the two randomness-generation protocols have different requirements. A common-coin scheme does not have to be always unpredictable and agreed-upon, but only with some constant probability [32, 11]. It should, however, be efficient, as it is used in every agreement instance within the broadcast protocol. On the other hand, the seed-generation protocol must always be unpredictable and agreed upon, but it can be slow, as it is only run periodically (e.g., once per epoch).

Related work. Multiple common-coin constructions without a trusted dealer have been proposed in the literature. Ben-Or [3] presents a simple protocol, where every party flips a *local coin.* As a result, parties agree on the value of the coin only with probability $\Theta(2^{-n})$, A common-coin scheme from *verifiable secret sharing* has been shown by Canetti and Rabin [11], but their resulting Byzantine agreement protocol has communication complexity $\mathcal{O}(n^{11})$. Patra, Choudhury, and Rangan [32] bring this down to $\mathcal{O}(n^3)$.

A different approach constructs common coins from *publicly verifiable secret sharing.* The resulting protocols, such as SCRAPE [12], ALBATROSS [13], Spurt [19], HydRand [37], and RandHound-

RandHerd [40], are efficient, yet they all make synchrony assumptions. RandShare [40] has been formalized in the asynchronous communication setting, but it is, according to its authors, less efficient.

Another line of work is based on *time-based cryptography.* Protocols in this category, such as Unicorn [30] and Bicorn [15], employ *verifiable delay functions* [6] and rely on the assumption that certain functions (such as exponentiation in groups of unknown order [36]) can only be computed serially. None of the aforementioned works explicitly mentions the network assumptions. Overviews of random beacon protocols are given by Raikwar and Gligoroski [35], and by Choi, Manoj, and Bonneau [16].

Multiple works that circumvent ADKG exist in the literature, but they either make more assumptions, have non-optimal resilience, or result in inefficient protocols. Existing PoS blockchains rely on the timely delivery of honestly generated blocks, hence make timing assumptions. Ouroboros Praos [21] implements a randomness beacon protocol, used as seed in their leader-election algorithm, by hashing a large number of VRF outputs. Partial-synchrony assumptions assure that the honestly generated VRF outputs cannot be delayed arbitrarily by the adversary. King and Saia [28, 29] propose a synchronous common-coin protocol that makes uses of pseudorandomly selected committees, but achieves non-optimal resilience. This is improved in the protocol of Algorand [26, 14], where each committee member applies a VRF on the seed of previous block, and then the smallest valid VRF value sent by some committee member is kept. The protocol is first described in the synchronous model [26] and later extended to the partially synchronous [14]. Cohen, Keidar, and Spiegelman [17] extend this idea to the asynchronous model, but their protocol achieves an $n = 4.5t$ resilience. In all these protocols the coins are not reusable and the whole coin-generation algorithm has to be run repeatedly.

Blum *et al.* [4] also generate randomness without ADKG. Their ATOB protocol works in the following way. Assume first that a trusted dealer publishes on a ledger all the setup material required for one instance of Byzantine agreement and one instance of a Multiparty Computation (MPC) protocol. Then, on every invocation of the agreement protocol, parties use the Byzantine-agreement setup in the agreement protocol and the MPC setup in a tailor-made MPC protocol that refreshes the whole setup. Finally, they replace the trusted dealer with a standard MPC protocol, executed once in a distributed setup phase. This blueprint solves the problem of dynamic stake elegantly, but, the proposed MPC protocol for refreshing the setup, which has to be executed for *every* Byzantine agreement instance, is not efficient: it employs Threshold Fully Homomorphic Encryption (TFHE), digital signatures, and zero-knowledge proofs.

Contributions. In this paper we address all the aforementioned limitations of randomness generation for the first time. We present asynchronous seed-generation and common-coin protocols that

- require no trusted setup,
- support optimal resilience $t < n/3$,
- employ small committees and are concretely efficient,
- directly support the PoS setting and dynamic participation,
- are modular and can be generically used in any ATOB broadcast.

Our methods. We are motivated by the question whether one can use the simple, practical, and efficient approach of getting common coins from threshold setup without running inefficient and complicated protocols whenever the stake has shifted. Building on the idea of Blum *et al.* [4], we rely on the fact that there already exists a functional ATOB: we generate the setup assuming that we already have the ATOB, and then use the generated setup to keep the ATOB running. To maintain practical efficiency the crucial step is to avoid FHE. We achieve this by generating weaker setups than Blum *et al.* [4], nonetheless still strong enough for the continued execution of the ATOB.

A crucial observation is that coins consumed by Byzantine agreement do not need to be perfect, i.e., always unpredictable and agreed upon [11, 32]. Hence, instead of generating a single, perfect threshold setup, we generate several candidate setups, such that some *constant* fraction of them are good. Many DKG protocols can be seen as doing this as their first step, but their next step is to combine them into a single perfect setup. In order to be combinable, the setups must be of a particular form, and the committee that holds the setup must be *good* (that is, contain less than a threshold corruptions) except with negligible probability. As our setups are not combined and our committees only need to be good with a constant probability, our protocols are simpler and more efficient, and use smaller committees.

Both seed, our seed-generation protocol, and wMDCF, our common-coin protocol, follow the approach depicted in Figure 1. They elect a *proposers committee*, each member of which is expected to create a *setup* (a *VSS setup* or a *coin setup*, for seed and wMDCF, respectively). Each elected proposer is assigned a *holding committee*, for which it creates the threshold setup. For this, the proposer acts as a dealer, encrypts the private setup material under the keys of the holding committee, and broadcasts these encryptions and the required verifications keys with a single message on the ATOB. We use a VRF-based lottery to determine both the proposers and the holding committees, where each party is elected with probability proportional to its stake. To open a seed value in the seed protocol, each of the holding committees reveals its shares and these are all added together. To flip a coin cid in the wMDCF protocol, we first hash cid with a seed to obtain a pointer to one of the published setups and then use that setup to obtain the value of cid. Which setup will be used for each cid is thus unpredictable until the seed is known.

Organization. The rest of this paper is organized as follows. Section 2.1 presents the formal model used in the schemes and Section 2.2 presents the primitives used in our schemes. Then, each of the seed-generation and common-coin protocols are presented in modular way, in two steps. Section 3 presents *wVSS*, a weak verifiable secret sharing scheme, which is then used in Section 4 to build the *seed-generation* protocol. Section 5 presents wHDCF, a weak honest-dealer coin-flip protocol, which is then used in Section 6 to build the wMDCF common-coin protocol. All of these schemes are parameterized over committee sizes and thresholds, and secure bounds for these are computed in Section 7. In Section 9 we show how to instantiate a concrete ATOB with our coins. In Section 8 we analyze the concrete communication cost of our protocols.

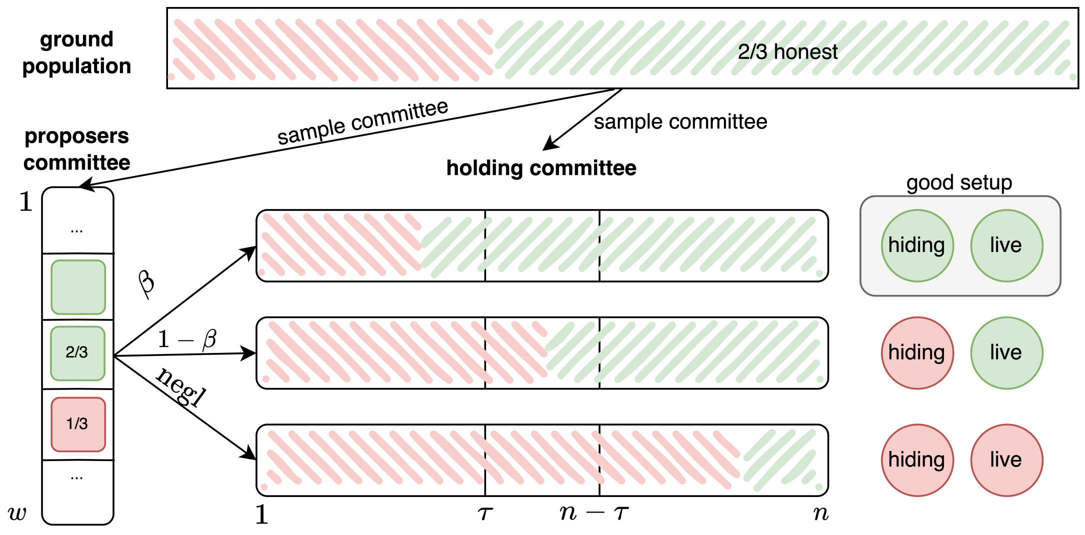

Figure 1: The high level idea of our protocols. A *proposers committee* is elected, and we wait until w proposers broadcast a setup. Assuming $2/3$ honesty in the ground population, a proposer is honest with probability $2/3$. Each proposer is assigned a *holding committee* of size n and creates an (n, τ) threshold setup for it. A committee is *hiding* if it contains at most τ corrupted parties, and *live* if it contains at most $n - \tau - 1$ corrupted parties. A setup is *good* if its proposer in honest and its holding committee is hiding and live. We set w so as to have enough honest proposers, and n and τ so that each holding committee is hiding with constant probability β and live with all but negligible probability. As a result, we get good setups with a constant probability γ.

2 Preliminaries

2.1 Model

We assume a model with asynchronous authenticated point-to-point channels. In addition we assume an asynchronous persistent total-order broadcast channel. We denote by Ledger the totally-ordered sequence of messages that have been delivered on the channel. We point out that if a blockchain has a distinction between final and non-final messages, then Ledger denotes the final messages. We assume that when a protocol is started all the parties taking part in the protocol agree on a session identifier sid and an existing point on the ledger, $p \le |\mathsf{Ledger}|$. We think of p as the *starting point of the protocol*, which gives consensus on the context of the protocol like stake distribution and lottery as discussed below. Protocols can have *public output* which might not be explicitly posted on the ledger, but will have a well-defined value and virtual point p at which they happened.

Definition 1 (Public output) We say that $\mathsf{PubOutF}$ is a public output function if it computes a public output from a ledger Ledger and a session identifier sid, where either $\mathsf{PubOutF}(\mathsf{Ledger}, \mathsf{sid}) = y \in \{0,1\}^*$ or $\mathsf{PubOutF}(\mathsf{Ledger}, \mathsf{sid}) = \perp$. We require that if $\mathsf{PubOutF}(\mathsf{Ledger}, \mathsf{sid}) \neq \perp$ then $\mathsf{PubOutF}(\mathsf{Ledger} \| m, \mathsf{sid}) = \mathsf{PubOutF}(\mathsf{Ledger}, \mathsf{sid})$ for all m. We say that sid gave public output y at position p if $|\mathsf{Ledger}| \ge p$ and $\mathsf{PubOutF}(\mathsf{Ledger}[1, p-1], \mathsf{sid}) = \perp$ and $\mathsf{PubOutF}(\mathsf{Ledger}[1, p], \mathsf{sid}) = y$. Unless multiple sid's are in scope we will omit the sid parameter. Finally we will informally say that some protocol gives public output $\mathsf{PubOutF}$ when additionally the ledger is implicit or when it is an eventual property of the ledger. □

Dynamic Stake. We consider proof-of-stake defined via the ledger. For each Ledger there is a stake distribution $\Sigma(\mathsf{Ledger}) : \mathbb{P} \to \mathbb{R}_0$ which may change as the ledger grows, can be computed in poly-time, and which gives for each party P its stake $\Sigma(\mathsf{Ledger})(\mathsf{P})$. For each point p there is also a stake distribution Σ_p, which is the stake distribution used by protocols with p as starting point. It may be different from $\Sigma(\mathsf{Ledger}[1, p])$, as discussed below.

Lotteries. In PoS based protocol it is common that parties are selected at random for carrying out a role in the protocol, like serving on a committee or producing the next block in a blockchain. To keep the model simple we assume that this is done via a random oracle. To keep the model simple we assume that for each point p on the ledger there is a random oracle $\Gamma_p : \{0,1\}^* \to \{0,1\}^\lambda$. We assume that Γ_p is sampled and made available to the parties at some point *after* Σ_p can be computed from Ledger. This ensure that Γ_p is independent of Σ_p. If Γ_p was made available before Σ_p was fixed then corrupted parties could update Σ_p based on Γ_p and for instance give more stake to parties "lucky" in Γ_p. One way to implement this is to iteratively generate random and unpredictable seeds seed appearing as public outputs. Then for a given point p let seed_p be the latest seed on $\mathsf{Ledger}[1, p]$, let $\Gamma_p(x) = R(\mathsf{seed}_p, x)$ for a random oracle R, let $p' < p$ be the latest point where seed was unpredictable, and let $\Sigma_p = \Sigma(\mathsf{Ledger}[1, p'])$.

Our protocols include steps where a party samples a committee cid of size n. We model this as a function $\mathsf{SampleCommittee}_p(\mathsf{cid}, n) \to \left(\mathsf{H}_i\right)_{i\in[n]}$ that uses Γ_p to sample n parties from $\mathbb{P}$ with probability proportional to the stake Σ_p. As the input is public, the output can be verified by a function $\mathsf{VerifyCommittee}_p(\mathsf{cid}, \left(\mathsf{H}_i\right)_{i\in[n]})$ that reruns $\mathsf{SampleCommittee}_p(\mathsf{cid}, n)$ and verifies that it matches $\left(\mathsf{H}_i\right)_{i\in[n]}$. We assume $\mathsf{SampleCommittee}_p(\mathsf{cid}, n)$ is locally computable by every party. Using our lottery abstraction this could be implemented by calling $\Gamma_p(\mathsf{cid}, i)$, for some committee cid and for $i \in [n]$, to obtain a number $r_i \in \{0, 2^\lambda - 1\}$, and then deterministically mapping r_i to a party $P_i \in \mathbb{P}$ based on Σ_p. Observe that a party with relatively large stake can appear multiple times in the committee.

2.2 Primitives

Our schemes make use of the following primitives.

2.2.1 Public-Key Encryption with Full Decryption

There are keys $(\mathsf{dk}_i, \mathsf{ek}_i)$, for all $\mathsf{P}_i \in \mathbb{P}$, for an IND-CPA encryption scheme with full decryption, PKE. Encrypting a message $m \in \mathsf{PKE}.\mathcal{M}$ using randomness $r \in \mathsf{PKE}.\mathcal{R}$ results in a ciphertext $c = \mathsf{Enc}_{\mathsf{ek}_i}(m; r) \in \mathsf{PKE}.\mathcal{C}$. Given a ciphertext $c \in \mathsf{PKE}.\mathcal{C}$ the decryption algorithm $\mathsf{Dec}_{\mathsf{dk}_i}(c)$ returns both $m \in \mathsf{PKE}.\mathcal{M}$ and $r \in \mathsf{PKE}.\mathcal{R}$. The triple (m, r, c) can then be verified by anyone holding ek_i by checking if $\mathsf{Enc}_{\mathsf{ek}_i}(m; r) = c$. Given an invalid ciphertext a zero knowledge proof that the ciphertext is invalid can be obtained using the secret key.

Construction using El Gamal. We first show that we can obtain the properties above in the random oracle model, as long as only encryptions of random messages are needed. This can then be lifted to a complete encryption scheme by symmetrically encrypting the message under a freshly sampled random key.

To encrypt a random value r, use El Gamal with $H(r)$ as randomness. I.e. if $\mathsf{dk} = x$ and $\mathsf{ek} = h = g^x$, then you encrypt r as $c = (A, B) = (g^{H(r)}, r \cdot h^{H(r)})$. To decrypt you first compute $r = B/(A^x)$, then check if re-encrypting using $H(r)$ as randomness gives back c. If verification checks out you can simply send r as proof. If the re-encryption does not match, you provide a proof that r was obtained by decrypting c. Note that (A, B) decrypts to r (under (g, h)) iff $\mathrm{DL}_g(h) = \mathrm{DL}_A(B/r)$, so this proof can be constructed using the Fiat-Shamir transform of the Σ-protocol for equality of discrete logarithms.

In the full scheme, in order to encrypt m using randomness r, you encrypt r as above and additionally include a symmetric encryption of m using r as key. To decrypt you first use regular El Gamal decryption to obtain r and verify it by re-encrypting. If it was encrypted correctly you use it to decrypt m and return (m, r), otherwise return $(\perp, r)$.

2.2.2 Weak Threshold Coin Flip

We use a (n, t)-threshold coin-flip (CF) scheme, where n is the total number of parties, t is the corruption threshold, and the reconstruction threshold is $t + 1$. The scheme has the following interface.

- $\mathsf{Setup}(n, t) \to (\mathsf{vk}, \mathsf{sk}_1, \ldots, \mathsf{sk}_n)$: The dealer generates a verification key vk and secret key shares sk_i of P_i. The secret keys can be used to create coin shares of multiple coins.
- $\mathsf{VerifyKeyShare}(\mathsf{vk}, i, \mathsf{sk}_i) \to b \in \{0, 1\}$: Given the verification keys vk, it verifies sk_i.
- $\mathsf{Flip}(\mathsf{sk}_i, \mathsf{coin}) \to (\mathsf{s}_i, \mathsf{w}_i)$: Given a coin identifier coin and secret key sk_i, it returns a coin share s_i for coin and potentially a correctness proof w_i, i.e., a proof that the coin share has been computed correctly using sk_i.
- $\mathsf{VerifyCoinShare}(\mathsf{vk}, \mathsf{coin}, \mathsf{s}_i, \mathsf{w}_i) \to b \in \{0, 1\}$: It verifies coin share s_i for coin identifier coin using the correctness proof w_i and verification key vk.
- $\mathsf{Combine}(\mathsf{coin}, \{\mathsf{s}_{i_j}\}_{j \in [t+1]}) \to \mathsf{s} \in \{0, 1\}^\lambda$: Given $t + 1$ valid coin shares s_{i_j}, for $j \in [t + 1]$, it returns the value s of the coin identifier coin.
- $\mathsf{VerifyCoin}(\mathsf{vk}, \mathsf{coin}, \mathsf{s}) \to b \in \{0, 1\}$: It verifies s as the value of coin identifier coin using the verification key vk.

Security properties. Assuming an honest dealer, i.e., that $\mathsf{Setup}()$ is correctly executed, and that there are no more than t corrupted parties, the scheme satisfies the following properties.

Completeness If the dealer is honest then all key shares generated with $\mathsf{Flip}(\mathsf{sk}_i, \mathsf{coin})$ will verify with $\mathsf{VerifyCoinShare}$.

Agreement For any $t + 1$ valid key shares the value $\mathsf{Combine}(\mathsf{coin}, \{\mathsf{s}_{i_j}\}_{j \in [t]})$ is the same, which define *the* value $\mathsf{s}_{\mathsf{coin}}$.

Unpredictability The value $\mathsf{s}_{\mathsf{coin}}$ is unpredictable without honest shares, i.e., for a set $C = \{\mathsf{P}_{i_j}\}_{j\in[t+1]}$ of corrupted parties, if a poly-time adversary has been given vk and sk_i for $\mathsf{P}_i \in C$ for a random setup and has not been given $\mathsf{Flip}(\mathsf{sk}_i, \mathsf{coin})$ for $\mathsf{P}_i \notin C$, then it cannot guess $\mathsf{s}_{\mathsf{coin}}$ better than at random. This holds even if it has access to an oracle giving $\mathsf{Flip}(\mathsf{sk}_i, \mathsf{coin}')$ for all honest P_i for all $\mathsf{coin}' \neq \mathsf{coin}$.

Instantiation. Scheme CF can be instantiated with any non-interactive unique threshold signature scheme, such as BLS threshold signatures [7, 5]. The dealer picks a random secret key sk and shares it among all n parties using a polynomial $\phi(X) = \sum_{k=0}^{t} \phi_k X^k$, such that $\phi_0 = \mathsf{sk}$. The only difference from threshold BLS is in $\mathsf{Setup}()$: it runs the key generation algorithm of the threshold signature scheme, but it does not return the verification keys in the form $g_2^{\mathsf{sk}_i} \in G_2$, where $i \in [n]$ and g_2 is the generator of G_2, as in the original scheme. Instead, it returns a vector $(V_0, \ldots, V_t)$, where $V_k = g^{\phi_k} \in G_2$, for $k \in \{0, \ldots, t\}$, i.e., it returns Feldman commitments [24] to the coefficients of ϕ. This allows us to implement $\mathsf{VerifyKeyShare}()$, so P_i can verify that its key share sk_i is indeed a point on polynomial ϕ by checking whether

$$g_2^{\mathsf{sk}_i} \stackrel{?}{=} \prod_{k=0}^{t} (V_k)^{i^k}. \qquad (1)$$

Observe that the original verification keys can still be obtained using (1) with input vk and i, hence $\mathsf{VerifyCoinShare}()$ and $\mathsf{VerifyCoin}()$ need no modification. Algorithm $\mathsf{Flip}()$ returns a signature share s_i on message coin using the key share sk_i of party P_i. Algorithm $\mathsf{Combine}()$ creates the threshold signature s from $t+1$ valid signature shares, which can then be hashed to get a value in $\{0, 1\}$. Algorithms $\mathsf{VerifyCoinShare}()$ and $\mathsf{VerifyCoin}()$ invoke the signature verification algorithm, which, in the case of BLS, only takes as input the message coin and a signatures share s_i or signature s, i.e., $\mathsf{w}_i = \perp$, and uses a pairing function. Alternatively, one can use the common-coin scheme of Cachin, Kursawe, and Shoup [9], but $\mathsf{VerifyCoin}()$ would additionally need as input the $t+1$ valid coin shares and proofs $\{\mathsf{s}_{i_j}, \mathsf{w}_{i_j}\}_{j\in[t+1]}$.

2.2.3 Secret sharing

Our construction requires a secret sharing scheme TSS with threshold t with the following interface.

1. $\mathsf{Share}(s; r) \to (s_1, \ldots, s_n)$: It shares a secret s using randomness r to n secret shares $(s_1, \ldots, s_n)$.
2. $\mathsf{Reconstruct}(\{s_{i_j}\}_{j=1}^{t}) \to s'$: Given t shares it reconstructs some secret s'.

The hiding property says that the joint distribution of t shares s_i is independent of s. We can instantiate TSS with Shamir's secret sharing scheme [38].

2.2.4 Digital Signature

Finally, there are keys $(\mathsf{sk}_{\mathsf{P}}, \mathsf{vk}_{\mathsf{P}})$, for all $\mathsf{P} \in \mathbb{P}$, for a digital signature scheme DS with unique signatures.

3 Weak Verifiable Secret Sharing

In this section we define a weak VSS protocol. It is weak in the sense that it is sometimes not hiding. But it is always binding and live (allows reconstruction). There is a designated dealer D, which is one of the participating parties. We assume D is given as part of session identifier, $\mathsf{sid} = (\mathsf{D}, \mathsf{sid}')$, and hence is known by all parties when the instance is created.

Syntax. The syntax of wVSS is as follows.

Commit On input $(\textsc{commit}, \mathsf{sid}, m)$ to D it starts running the commitment protocol and may as a result help produce a public output (see Definition 1) $\mathsf{PubOutVSSCommit}$. On input $(\textsc{commit}, \mathsf{sid})$ to a participating party $\mathsf{P} \neq \mathsf{D}$ it starts running the commitment protocol and may as a result help produce a public output $\mathsf{PubOutVSSCommit}$.

Open On input $(\textsc{open}, \mathsf{sid})$ after $\mathsf{PubOutVSSCommit}(\mathsf{sid}, \mathsf{Ledger}_{\mathsf{P}}) \neq \bot$, a party P starts running the open protocol and may as a result output $(\textsc{done-open}, \mathsf{sid}, m_{\mathsf{P}}, \pi)$, where π is a proof that m_{P} is the output. The proof can be checked by any party P' for which $\mathsf{PubOutVSSCommit}(\mathsf{sid}, \mathsf{Ledger}_{\mathsf{P}'}) \neq \bot$ using $\mathsf{wVSSVerify}(\pi, m)$.

Security. The security properties of wVSS are as follows.

Termination (1) If D is honest and gets input $(\textsc{commit}, \mathsf{sid}, m)$, and all other honest parties get input $(\textsc{commit}, \mathsf{sid})$ then eventually there is a public output $\mathsf{PubOutVSSCommit}$.

(2) If $\mathsf{PubOutVSSCommit}$ occurred and all honest parties get input $(\textsc{open}, \mathsf{sid})$ then eventually all honest parties give an output $(\textsc{done-open}, \mathsf{sid}, \cdot)$.

(3) If any honest party gives an output $(\textsc{done-open}, \mathsf{sid}, \cdot)$ then eventually all honest parties give output $(\textsc{done-open}, \mathsf{sid}, \cdot)$.

Validity If D is honest and had input $(\textsc{commit}, \mathsf{sid}, m)$ and some honest P gave output $(\textsc{done-open}, \mathsf{sid}, m_{\mathsf{P}}, \pi)$, then $m_{\mathsf{P}} = m$ and $\forall m' : \mathsf{wVSSVerify}(\pi, m') = \top \Leftrightarrow m' = m$.

Binding If D is corrupted, then the following holds. When the first honest party observes public output $\mathsf{PubOutVSSCommit}$ then one can in poly-time compute from the view of the adversary up to this point, a message m such that if later an honest party P gives output $(\textsc{done-open}, \mathsf{sid}, m_{\mathsf{P}}, \pi)$ then $m_{\mathsf{P}} = m$ and $\forall m' : \mathsf{wVSSVerify}(\pi, m') = \top \Leftrightarrow m' = m$.

β-Weak Hiding If D is honest then for each session sid it holds with probability $\beta > 0$ at a point in time t before any honest party got input $(\textsc{open}, \mathsf{sid})$ that m is hidden in the view of the adversary at time t, i.e., if $m = m_b$ for (m_0, m_1) picked by the adversary and a uniformly random bit b, then the adversary cannot guess b better than at random. A precise definition and security game is given with the analysis in Definition 3.

Construction. The central idea of our wVSS construction is to have a dealer choose a secret seed σ and secret share it unto a random holding committee and put on the ledger an encryption of each of the shares under the public key of the holder, this vector of encryptions is called the setup. If this was done correctly any $t + 1$ honest parties can decrypt their shares and use them to reconstruct σ. All randomness for the secret sharing and the encryption will be generated from σ

using a PRG. This allows the committee to rerun the setup procedure and check consistency with the published setup after reconstruction. This ensures that if anyone $t+1$ parties can reconstruct to some value σ, then all shares are correct, and therefore all subsets of $t+1$ shares reconstruct to the same σ. We also use randomness derived from σ to encrypt m and include the encryption in the setup. We cannot let the dealer pick the holding committee as we need enough honest parties on it to avoid deadlock of reconstruction. Therefore the holding committee is sampled pseudorandomly from the session identifier sid.

We use n_{VSS} and τ_{VSS} to define the size of the holding committee sampled by the dealer, and the reconstruction threshold in the holding committee, respectively. To ensure weak hiding these parameters should be chosen such that the sampled committee has at most τ_{VSS} corruptions with constant probability at least β, and to ensure liveness less than $n_{\text{VSS}} - \tau_{\text{VSS}}$ should be corrupted except with negligible probability. The scheme makes use of a signature scheme DS (Section 2.2.4), an encryption scheme PKE with full decryption (Section 2.2.1), a threshold secret-sharing scheme TSS (Section 2.2.3), a pseudorandom generator PRG, and a hash function H modelled as a random oracle.

Algorithm 1 Scheme wVSS, algorithm Commit, where an instance sid of wVSS is created at point p on Ledger. Code for process P_i.

```
1: function commit_value($\sigma$)
2:     $\rho = \mathsf{PRG}(H(\sigma))$
3:     $m_{mask} \overset{\$\rho}{\leftarrow} \{0,1\}^\lambda$
4:     $(s_1, \ldots, s_{n_{\text{vss}}}) \overset{\$\rho}{\leftarrow} \mathsf{TSS.Share}(\sigma)$
5:     for $j \in [n_{\text{VSS}}]$ do
6:         $r_j \overset{\$\rho}{\leftarrow} \{0,1\}^\lambda$;   $e_j \leftarrow \mathsf{PKE.Enc}_{\mathsf{ek}_j}(s_j, r_j)$
7:     return $((e_1, \ldots, e_{n_{\text{vss}}}), m_{mask})$

8: upon input $(\text{COMMIT}, \mathsf{sid}, \pi, m)$ where $\mathsf{sid} = (\mathsf{D}, \mathsf{sid}')$ and $\mathsf{P}_i = \mathsf{D}$ do      // only dealer D
9:     $(\mathsf{H}_1, \ldots, \mathsf{H}_{n_{\text{vss}}}) \leftarrow \mathsf{SampleCommittee}_p(\mathsf{sid}, n_{\text{VSS}})$
10:    $\sigma \leftarrow \mathsf{DS.Sign}_{\mathsf{sk_D}}(\mathsf{sid})$
11:    $((e_1, \ldots, e_{n_{\text{vss}}}), m_{mask}) \leftarrow \mathsf{commit\_value}(\sigma)$
12:    broadcast $(\mathsf{sid}, \pi, (e_1, \mathsf{H}_1) \ldots, (e_{n_{\text{vss}}}, \mathsf{H}_{n_{\text{vss}}}), m \oplus m_{mask})$ on Ledger
```

We implement Commit in Algorithm 1. In order to commit to a chosen value m, D first pseudorandomly samples a holding committee of size n_{VSS} (line 9). We say that the committee is "assigned" to D, as D cannot influence it without getting rejected as public output. The dealer then computes a signature σ on sid, obtaining a unique and unpredictable value. Then D commits to σ by secret-sharing it to the committee. This logic is extracted in an auxiliary function $\mathsf{commit\_value}$. It computes a random tape $\rho = \mathsf{PRG}(H(\sigma))$. This random tape is used in all subsequent steps that require randomness. Specifically, in line 3 a random message m_{mask} is sampled, in line 4 the value σ is secret-shared to the members of the holding committee using an $(n_{\text{VSS}}, \tau_{\text{VSS}})$-$\mathsf{TSS}$, and in lines 5– 6 the shares of σ are encrypted to the committee members. Each of these values are sampled *pairwise independently* from ρ. Finally, D broadcasts its *VSS setup* on Ledger (line 12). This VSS setup serves as a public output signalling that the message is committed and can at this point only be opened to some unique value–which could be $\perp$. We define the function $\mathsf{PubOutVSSCommit}(\mathsf{Ledger}, (\mathsf{D}, \mathsf{sid}'))$ as the earliest (in Ledger) message $\big((\mathsf{D}, \mathsf{sid}'), \pi, (e_1, \mathsf{H}_1) \ldots, (e_{n_{\text{vss}}}, \mathsf{H}_{n_{\text{vss}}}), m\big)$ which is signed by

D, where $\mathsf{VerifyCommittee}((\mathsf{D}, \mathsf{sid}'), \mathsf{H}_1, \ldots, \mathsf{H}_{n_{\text{vss}}}) = 1$. If no such message exists in Ledger, then $\mathsf{PubOutVSSCommit}(\mathsf{Ledger}, \mathsf{sid}) = \bot$.

Algorithm 2 Scheme wVSS, algorithm Open, where an instance sid of wVSS is created at point p on Ledger. Code only for process P_i is in the committee of instance sid, i.e., P_i is one of the H_j in the VSS-setup $(\mathsf{sid}, (e_1, \mathsf{H}_1) \ldots, (e_{n_{\text{vss}}}, \mathsf{H}_{n_{\text{vss}}}), m_{masked})$ published on Ledger.

```
    State:
13:     $\mathsf{validShares}[\mathsf{sid}] \leftarrow [\,]$

14: upon input $(\textsc{open}, \mathsf{sid})$ such that $\mathsf{PubOutVSSCommit}(\mathsf{Ledger}_{\mathsf{P}_i}, \mathsf{sid}) \neq \bot$ do
15:     let $((e_1, \mathsf{H}_1) \ldots, (e_{n_{\text{vss}}}, \mathsf{H}_{n_{\text{vss}}}), m_{masked}) = \mathsf{PubOutVSSCommit}(\mathsf{Ledger}_{\mathsf{P}_i}, \mathsf{sid})$
16:     let $\mathcal{C} = \{\mathsf{H}_1, \ldots, \mathsf{H}_{n_{\text{vss}}}\}$
17:     $(s'_i, r'_i) \leftarrow \mathsf{Dec}_{\mathsf{dk}_i}(e_i)$
18:     $e'_i \leftarrow \mathsf{Enc}_{\mathsf{ek}_i}(s'_i, r'_i)$
19:     if $e'_i = e_i$ then
20:         send $(\textsc{share}, s'_i, r'_i)$ to parties in $\mathcal{C}$
21:     else
22:         create zk-proof $\mathsf{W}_i$ that $e_i$ decrypts to $(s'_i, r'_i)$
23:         send $(\textsc{complaintEncryption}, \mathsf{sid}, \mathsf{W}_i, s'_i, r'_i)$ to parties in $\mathcal{C}$

24: upon deliver $(\textsc{share}, s_j, r_j)$ from $\mathsf{P}_j$ do
25:     if $e_j = \mathsf{Enc}_{\mathsf{ek}_j}(s_j, r_j)$ then
26:         append $s_j$ to $\mathsf{validShares}[\mathsf{sid}]$

27: upon $|\mathsf{validShares}[\mathsf{sid}]| = \tau_{\text{vss}} + 1$ do
28:     let $\left(s_{j_1}, \ldots, s_{j_{\tau_{\text{vss}}+1}}\right) = \mathsf{validShares}[\mathsf{sid}]$
29:     $\sigma' \leftarrow \mathsf{TSS.Reconstruct}(\{s_{j_k}\}_{j \in [\tau_{\text{vss}}+1]})$
30:     if $\mathsf{DS.Ver}_{\mathsf{vk}_\mathsf{D}}(\sigma', \mathsf{sid}) = 0$ then
31:         output $(\textsc{done-open}, \mathsf{sid}, \bot, \mathsf{validShares}[\mathsf{sid}])$
32:     $\left((e'_1, \ldots, e'_{n_{\text{vss}}}), m_{mask}\right) \leftarrow \mathsf{commit\_value}(\sigma')$
33:     if $(e'_1, \ldots, e'_{n_{\text{vss}}}) \neq (e_1, \ldots, e_{n_{\text{vss}}})$ then
34:         output $(\textsc{done-open}, \mathsf{sid}, \bot, \mathsf{validShares}[\mathsf{sid}])$
35:     else
36:         output $(\textsc{done-open}, \mathsf{sid}, m_{mask} \oplus m_{masked}, \sigma')$

37: upon deliver $c = (\textsc{complaintEncryption}, \mathsf{sid}, \mathsf{W}_j, s_j, r_j)$ do
38:     $e'_j \leftarrow \mathsf{PKE.Enc}_{\mathsf{ek}_j}((s_j, r_j))$
39:     if $e'_j \neq e_j$ and $\mathsf{W}_j$ is valid then
40:         output $(\textsc{done-open}, \mathsf{sid}, \bot, c)$
```

We implement Open in Algorithm 2. On input OPEN, and after $\mathsf{PubOutVSSCommit}(\mathsf{Ledger}, \mathsf{sid}) \neq \bot$, a party in the holding committee of the sid instance parses the output as a VSS setup. Party P_i then decrypts e_i to get its share and the randomness used for encryption, (s_i, r_i) (line 17). It then re-encrypts (s_i, r_i) (line 18) to verify that the encryption was done correctly (line 19). If the encryption is valid they send (s_i, r_i) to the other parties, otherwise it sends a verifiable complaint (lines 22–23). The complaint includes a zero-knowledge proof that e_i decrypts to (s'_i, r'_i).

Upon receiving a share from P_j (line 24), party P_i verifies that the share sent by P_j indeed corresponds to the value e_j published on Ledger. If this is the case, the share is considered valid.

Observe that the share the dealer created for P_j might be wrong in the first place. This is detected upon reconstruction. Specifically, once $\tau_{\text{VSS}} + 1$ valid shares are received (line 27), P_i runs the reconstruction of TSS to get back some σ', which should be a signature on sid computed by D. Party P_i first verifies the signature and, if valid, it repeats the steps performed by D to secret-share σ' (line 32). Observe that, given σ', all steps in $\mathsf{commit\_value}()$ are deterministic. Hence, if the reconstructed σ' is the same as the σ computed by the dealer, then $\mathsf{commit\_value}(\sigma')$ will return the same values as the ones posted on Ledger by D or we detect that the dealer cheated and output $\perp$. This is checked in line 33. Finally, upon delivering a complaint, sent by some party P_j, party P_i verifies the complaint and, if valid, outputs $\perp$. Note that no party can produce a valid complaint if the check in line 33 goes through. The $\mathsf{wVSSVerify}$ check can simply be implemented by a function that when given an encryption complaint checks if it is valid as in line 37, and when given a set of shares checks that they are all valid and treats them as input to the activation rule in line 27 to see that the same output is obtained. When the output of wVSS needs to be distributed to the full set of parties, each party on the committee simply forwards their $(\textsc{done-open}, \mathsf{sid}, m_{\mathsf{P}}, \pi)$ message to the remaining parties. Note that even though the proof of the outputs can differ, an outside party only needs to receive one. Hence, in gossiping networks the output messages can be deduplicated by only forwarding the first valid one to lower communication complexity.

4 Generating an Unpredictable Seed

In this section we define a seed-generation protocol seed. A seed can be thought of as a perfect coin flip: there is agreement on the output and its value is unpredictable before the protocol starts.

Syntax. The syntax of seed is as follows:

Commit On input $(\textsc{seed}, \mathsf{sid})$ in a session with session identifier sid a party starts running the commit protocol and may as a result public output $\mathsf{PubOutSeedCommit}$.

Open On input $(\textsc{seed-open}, \mathsf{sid})$, which must be given after public output $\mathsf{PubOutSeedCommit}$, in a session with id sid a party starts running the opening protocol and may as a result output $(\textsc{done-seed}, \mathsf{sid}, c)$, for $c \in \{0,1\}^\lambda$.

Security. The security properties of seed are as follows:

Termination If all honest parties get inputs $(\textsc{seed}, \mathsf{sid})$ then eventually all honest parties get public output $\mathsf{PubOutSeedCommit}$.

If all honest parties get correct inputs $(\textsc{seed-open}, \mathsf{sid})$ then eventually all honest parties give an output $(\textsc{done-seed}, \mathsf{sid}, \cdot)$.

Agreement If two honest parties have outputs $(\textsc{done-seed}, \mathsf{sid}, c_{\mathsf{P}})$ and $(\textsc{done-seed}, \mathsf{sid}, c_{\mathsf{Q}})$ then $c_{\mathsf{P}} = c_{\mathsf{Q}}$. Call the common value c_{sid}.

Unpredictability For each session sid it holds that c_{sid} is unpredictable before the first honest party gets input $(\textsc{seed-open}, \mathsf{sid})$.

Algorithm 3 Scheme seed, algorithm Commit, where an instance sid of seed is created at some point p on Ledger. Code for process P_i.

```
41: upon input $(\textsc{seed}, \mathsf{sid})$ do
42:     $\mathcal{C} \leftarrow \mathsf{SampleCommittee}(\mathsf{sid}, m_{\textsc{seed}})$
43:     for $j \in [m_{\textsc{seed}}]$ such that $\mathcal{C}[j] = P_i$ do
44:         $r \overset{\$}{\leftarrow} \{0,1\}^{\lambda}$
45:         $\mathsf{wVSS}(\textsc{commit}, ((\mathsf{P}_i, j), \mathsf{sid}), r)$
```

Algorithm 4 Scheme seed, algorithm Open, where an instance sid of seed is created at some point p on Ledger. Code for process P_i.

```
46: State:
47:     $\mathsf{openings}[\mathsf{sid}] \leftarrow [\,]$

48: upon input $(\textsc{seed-open}, \mathsf{sid})$ such that $\mathsf{PubOutSeedCommit}(\mathsf{sid}, \mathsf{Ledger}) \neq \bot$ do
49:     $\mathsf{setups} \leftarrow \mathsf{PubOutSeedCommit}(\mathsf{sid}, \mathsf{Ledger})$
50:     for $j \in [w_{\textsc{seed}}]$ do
51:         $\mathsf{sid}_j \leftarrow \mathsf{setups}[j]$
52:         $\mathsf{wVSS}(\textsc{open}, \mathsf{sid}_j)$

53: upon deliver $(\textsc{done-open}, \mathsf{sid}_j, r, \pi)$ do
54:     if $j \in [w_{\textsc{seed}}] \wedge \mathsf{wVSSVerify}(\pi, r)$ then
55:         append $m$ to $\mathsf{openings}[\mathsf{sid}]$

56: upon $|\mathsf{openings}| = w_{\textsc{seed}}$
57:     output $(\textsc{done-seed}, \mathsf{sid}, \bigoplus_{r \in \mathsf{openings}[\mathsf{sid}]} r)$
```

Construction. The protocol uses parameters m_{SEED} and w_{SEED}. The idea is to sample m_{SEED} parties in $\mathbb{P}$ to contribute a wVSS setup, asynchronously wait for the first w_{SEED} setups and use the XOR of them as a seed. We discuss in Section 7 how to set these parameters, such that at least one good setup (that is, from an honest proposer and with a committee with at most τ_{VSS} corrupted members) appears on the ledger, except with negligible probability.

The protocol is started at some starting point p of Ledger, with associated stake Σ_p and committee sampling mechanism $\mathsf{SampleCommittee}_p()$. We implement Commit in Algorihm 3 and Open in Algorihm 4. A party that is elected to contribute a wVSS setup (line 42) picks a random r and starts an instance of wVSS to share r (lines 44–45). Once w_{SEED} wVSS protocols with session identifiers $\mathsf{sid}_j = (\mathsf{P}_j, k, \mathsf{sid})$, where $\mathcal{C}[k] = \mathsf{P}_j$, have given public output $\mathsf{PubOutVSSCommit}$ on Ledger, then we define $\mathsf{PubOutSeedCommit}$ to be the ordered tuple of the session identifiers of the first w_{SEED} such outputs. After this point, the value of the nonce is implicitly defined by the state of the ledger, and on input $(\text{SEED-OPEN}, \mathsf{sid})$, parties start running the Open algorithm on these w_{SEED} instances of wVSS (lines 48–52). By design, the holding committee of each of these instances has enough honest members for wVSS to terminate. The final seed value is defined as the XOR of the values output by each Open (line 57).

5 Weak Honest-Dealer Coin-Flip

In this section we define the weak honest-dealer coin-flip (wHDCF) protocol. In wHDCF there is a designated dealer D, which is one of the participating parties. We assume D is given as part of session identifier, $\mathsf{sid} = (\mathsf{D}, \mathsf{sid}')$, and hence is known by all parties when the instance is created. The scheme is *weak* in the sense that parties may output $\perp$ as the value of the coin, but if two honest parties output a value in $\{0,1\}$, then it will be the same. It is *honest-dealer* as the coin value becomes predictable for a corrupted D. The scheme makes use of a committee verification mechanism $\mathsf{SampleCommittee}_p()$ proportional to stake at point p (Section 2.1), an encryption scheme with full decryption PKE (Section 2.2.1), and an (n_{COIN}, τ_{COIN})-threshold weak coin flip scheme CF (Section 2.2.2). Here n_{COIN} and τ_{COIN} are protocol parameters, for which we choose specific values in Section 7.

Syntax. The syntax of weak honest-dealer coin-flip is as follows:

Deal On input $(\text{DEAL}, \mathsf{sid})$ a participating party starts running the dealing protocol of CF and may as a result produce a public output $\mathsf{PubOutSingleDeal}$.

Flip On input $(\text{FLIP}, \mathsf{sid}, \mathsf{cid})$, for coin identifier cid, after $\mathsf{PubOutSingleDeal}(\mathsf{sid}, \mathsf{Ledger}) \neq \perp$, a party starts running the flip protocol of CF and outputs $(\text{DONE-FLIP}, \mathsf{sid}, \mathsf{cid}, s, \pi)$, where $s \in \{\perp\} \cup \{0,1\}^\lambda$ and π is a proof that s is the output of the coinflipping protocol. The proof can be checked by any party P' for which $\mathsf{PubOutSingleDeal}(\mathsf{sid}, \mathsf{Ledger}_{\mathsf{P}'}) \neq \perp$ using $\mathsf{wHDCFVerify}(\pi, m)$.

Security. The security properties of wHDCF are as follows.

Termination (1) If D is honest and all honest parties get input $(\text{DEAL}, \mathsf{sid})$, then eventually $\mathsf{PubOutSingleDeal}(\mathsf{sid}, \mathsf{Ledger}) \neq \perp$.

(2) If, after $\mathsf{PubOutSingleDeal}(\mathsf{sid}, \mathsf{Ledger}) \neq \bot$, all honest parties get input $(\textsc{flip}, \mathsf{sid}, \mathsf{cid})$, then eventually all honest parties give output $(\textsc{done-flip}, \mathsf{sid}, \mathsf{cid}, \cdot)$, except with negligible probability.

Weak Agreement If two honest parties output $(\textsc{done-flip}, \mathsf{sid}, \mathsf{cid}, c_\mathsf{P}, \pi)$ and $(\textsc{done-flip}, \mathsf{sid}, \mathsf{cid}, c_\mathsf{Q}, \pi)$, such that $c_\mathsf{P} \neq \bot$ and $c_\mathsf{Q} \neq \bot$, then $c_\mathsf{P} = c_\mathsf{Q}$, except with negligible probability. The same holds if $c_\mathsf{Q} \neq \bot$ and $\mathsf{wHDCFVerify}(\pi, c_\mathsf{Q}) \neq \bot$. Moreover, if D is honest, then no honesty party P outputs $c_\mathsf{P} = \bot$.

Honest-Dealer β-Unpredictability If dealer D of session sid is honest, then each coin flip cid is independently unpredictable with some constant probability $\beta > 0$, where β is defined when $\mathsf{PubOutSingleDeal}(\mathsf{sid}, \mathsf{Ledger}) \neq \bot$ and is independent of cid.

A more formal version of *Honest-Dealer β-Unpredictability* is given in Definition 5 and the proofs in Appendix C.

Construction. In a high level, the scheme works as follows. Dealer D is assigned a *coin-holding committee* of size n_COIN and creates a *coin setup* for an (n_COIN, τ_COIN)-threshold coin scheme CF for this committee. Termination is achieved by appropriately setting the parameters and from the pseudorandom nature of the committee: if the dealer completes the setup, there are at least $\tau_\text{COIN}+1$ honest parties in the committee, except with negligible probability. The weak agreement property is achieved by *verifiable* complaints against a corrupted dealer. Upon receiving a complaint valid complaint, a party terminates the Flip protocol outputting $\bot$. If, additionally, D is honest, then our protocol guarantees unpredictability with constant probability β, defined as the probability of having at most τ_COIN corruptions in the committee, and depending only on n_COIN and τ_COIN.

Algorithm 5 Scheme wHDCF, algorithm Deal, where an instance sid of wHDCF is created at point p on Ledger. Code for process P_i.

```
58: upon input $(\textsc{deal}, \mathsf{sid})$ where $\mathsf{sid} = (\mathsf{D}, \mathsf{sid}')$ and $\mathsf{P}_i = \mathsf{D}$ do        // only dealer D
59:     $(\mathsf{H}_1, \ldots, \mathsf{H}_{n_\text{COIN}}) \leftarrow \mathsf{SampleCommittee}_p(\mathsf{sid}, n_\text{COIN})$,
60:     $(\mathsf{vk}, \mathsf{sk}_1, \ldots, \mathsf{sk}_{n_\text{COIN}}) \leftarrow \mathsf{CF.Setup}(n_\text{COIN}, \tau_\text{COIN})$
61:     for $j \in [n_\text{COIN}]$ do
62:         $r_j \xleftarrow{\$} \{0,1\}^\lambda$;   $e_j = \mathsf{PKE.Enc}_{\mathsf{ek}_j}((\mathsf{sk}_j, r_j))$
63:     broadcast $(\mathsf{sid}, \mathsf{vk}, (\mathsf{H}_1, e_1), \ldots, (\mathsf{H}_{n_\text{COIN}}, e_{n_\text{COIN}}))$ on Ledger
```

In Algorithm 5 we implement Deal. The dealer first (line 59) samples the coin-holding committee of size n_COIN and then (line 60) uses CF to create a coin setup for it. The coin setup includes secret keys $\mathsf{sk}_1, \ldots, \mathsf{sk}_{n_\text{COIN}}$ and verification key vk. Each secret key sk_i is encrypted to party's P_i long term private key ek_i using a fresh randomness r_i (lines 61–62). The coin setup is broadcast on Ledger. When a coin-setup is included in Ledger we define the public output $\mathsf{PubOutSingleDeal}(\mathsf{sid}, \mathsf{Ledger})$ as $(\mathsf{vk}, (\mathsf{H}_1, e_1), \ldots, (\mathsf{H}_{n_\text{COIN}}, e_{n_\text{COIN}}))$ if the included committee verifies using $\mathsf{VerifyCommittee}$. Otherwise the output is $\bot$.

In Algorithm 6 we implement Flip. Only parties in the coin-holding committee run it. When P_i gets input $(\textsc{flip}, \mathsf{sid}, \mathsf{cid})$ and $\mathsf{PubOutSingleDeal}(\mathsf{sid}, \mathsf{Ledger}_{\mathsf{P}_i}) \neq \bot$, it first reads the coin setup and tries to decrypt e_i to obtain its key share (line 70). Scheme PKE returns sk_i' and the randomness r_i' that D is supposed to have used at encryption time. Party P_i checks whether D has indeed

Algorithm 6 Scheme wHDCF, algorithm Flip (cid), where an instance sid of wHDCF is created at point p on Ledger. Code for process P_i, P_i is one of the H_j in the coin-setup $(\mathsf{sid}, \mathsf{vk}, (\mathsf{H}_1, e_1), \dots, (\mathsf{H}_{n_{\text{COIN}}}, e_{n_{\text{COIN}}}))$ published on Ledger.

```
    State:
64:     $\mathsf{validShares}[\mathsf{sid}][\mathsf{cid}] \leftarrow [\,]$, for each $\mathsf{sid}$ and $\mathsf{cid}$
65:     $\mathsf{justifiedComplaint}[\mathsf{sid}][\mathsf{cid}] \leftarrow \bot$, for each $\mathsf{sid}$ and $\mathsf{cid}$
66:     $\mathsf{terminated}[\mathsf{sid}][\mathsf{cid}] \leftarrow 0$, for each $\mathsf{sid}$ and $\mathsf{cid}$

67: upon input $(\text{FLIP}, \mathsf{sid}, \mathsf{cid})$ such that $\mathsf{PubOutSingleDeal}(\mathsf{sid}, \mathsf{Ledger}) \neq \bot$ do
68:     $(\mathsf{vk}, (\mathsf{H}_1, e_1), \dots, (\mathsf{H}_{n_{\text{COIN}}}, e_{n_{\text{COIN}}})) \leftarrow \mathsf{PubOutSingleDeal}(\mathsf{sid}, \mathsf{Ledger})$
69:     let $\mathcal{C} = \{\mathsf{H}_1, \dots, \mathsf{H}_{n_{\text{COIN}}}\}$
70:     $(\mathsf{sk}'_i, r'_i) = \mathsf{PKE.Dec}_{\mathsf{dk}_i}(e_i)$
71:     if $e_i \neq \mathsf{PKE.Enc}_{\mathsf{ek}_i}((\mathsf{sk}'_i, r'_i))$ then
72:         create zk-proof $\mathsf{W}_i$ that $e_i$ decrypts to $(\mathsf{sk}'_i, r'_i)$
73:         send $(\text{COMPLAINTENCRYPTION}, \mathsf{sid}, \mathsf{cid}, \mathsf{W}_i, \mathsf{sk}'_i, r'_i)$ to parties in $\mathcal{C}$; return
74:     if $\mathsf{CF.VerifyKeyShare}(\mathsf{vk}, i, \mathsf{sk}'_i) = 0$ then
75:         send $(\text{COMPLAINTKEYSHARE}, \mathsf{sid}, \mathsf{cid}, \mathsf{sk}'_i, r'_i)$ to parties in $\mathcal{C}$; return
76:     $\mathsf{s}_i = \mathsf{CF.CreateShare}(\mathsf{sk}'_i, \mathsf{cid})$
77:     send $(\text{COINSHARE}, \mathsf{sid}, \mathsf{cid}, \mathsf{s}_i)$ to parties in $\mathcal{C}$

78: upon deliver $(\text{COINSHARE}, \mathsf{sid}, \mathsf{cid}, s_j)$ from $\mathsf{P}_j$ do
79:     if $\mathsf{CF.VerifyCoinShare}(\mathsf{vk}, \mathsf{cid}, s_j) = 1$ then
80:         append $s_j$ to $\mathsf{validShares}[\mathsf{sid}][\mathsf{cid}]$

81: upon deliver $c = (\text{COMPLAINTENCRYPTION}, \mathsf{sid}, \mathsf{cid}, \mathsf{W}_j, \mathsf{sk}_j, r_j)$ do
82:     $e'_j \leftarrow \mathsf{PKE.Enc}_{\mathsf{ek}_j}((\mathsf{sk}_j, r_j))$
83:     if $e'_j \neq e_j$ and $\mathsf{W}_j$ is valid then
84:         $\mathsf{justifiedComplaint}[\mathsf{sid}][\mathsf{cid}] \leftarrow c$

85: upon deliver $c = (\text{COMPLAINTKEYSHARE}, \mathsf{sid}, \mathsf{cid}, \mathsf{sk}_j, r_j)$ do
86:     $e'_j \leftarrow \mathsf{PKE.Enc}_{\mathsf{ek}_j}((\mathsf{sk}_j, r_j))$
87:     if $e'_j = e_j$ and $\mathsf{CF.VerifyKeyShare}(\mathsf{vk}, i, \mathsf{sk}_i) = 0$ then
88:         $\mathsf{justifiedComplaint}[\mathsf{sid}][\mathsf{cid}] \leftarrow c$

89: upon $|\mathsf{validShares}[\mathsf{sid}][\mathsf{cid}]| = \tau_{\text{COIN}} + 1$ and $\mathsf{terminated}[\mathsf{sid}][\mathsf{cid}] = 0$ do
90:     let $\left(s_{j_1}, \dots, s_{j_{\tau_{\text{COIN}}+1}}\right) = \mathsf{validShares}[\mathsf{sid}][\mathsf{cid}]$
91:     $s \leftarrow \mathsf{CF.Combine}(\mathsf{cid}, \{s_{j_k}\}_{k \in [\tau_{\text{COIN}}+1]})$
92:     $\mathsf{terminated}[\mathsf{sid}][\mathsf{cid}] \leftarrow 1$
93:     output $(\text{DONE-FLIP}, \mathsf{sid}, \mathsf{cid}, s, \mathsf{validShares}[\mathsf{sid}][\mathsf{cid}])$

94: upon $\mathsf{justifiedComplaint}[\mathsf{sid}][\mathsf{cid}] \neq \bot$ and $\mathsf{terminated}[\mathsf{sid}][\mathsf{cid}] = 0$ do
95:     $\mathsf{terminated}[\mathsf{sid}][\mathsf{cid}] \leftarrow 1$
96:     output $(\text{DONE-FLIP}, \mathsf{sid}, \mathsf{cid}, \bot, \mathsf{justifiedComplaint}[\mathsf{sid}][\mathsf{cid}])$
```

done so by re-encrypting (sk'_i, r'_i) and checking the result against e_i. If it is different, P_i sends a COMPLAINTENCRYPTION message that includes a zero-knowledge proof that e_i decrypts to (sk'_i, r'_i) (lines 71–73) and stops handling the Flip event. Otherwise, P_i can prove correct decryption of e_i in a complaint message by sending (sk'_i, r'_i). Party P_i then verifies its key share against the verification vector vk published in the coin setup, and, if it is invalid, sends a COMPLAINTKEYSHARE message to $\mathcal{C}$ (lines 74–75) and returns. If the check passes, it creates a coin share using the threshold-coin scheme CF (line 76) and sends to the committee $\mathcal{C}$. All complaints are verifiable: COMPLAINTENCRYPTION is valid if the zk-proof W_j, proving that the published e_j decrypts to $(\mathsf{sk}_j, r_j))$, is valid, and the re-encryption of (sk_j, r_j) gives something different from e_j (lines 81–84). COMPLAINTSHARE is valid if the re-encryption of (sk_j, r_j) gives the published e_j and the key share sk_j is deemed invalid by the CF scheme. (lines 85–88). Party P_i outputs in two cases, whichever comes first. First, upon collecting $\tau_{\text{COIN}} + 1$ valid coin shares (line 89), in which case the value of the coin is reconstructed using the underlying CF scheme. Second, upon receiving a valid complaint (line 94), in which case a $\perp$ value is output. The wHDCFVerify check can be implemented by a function that when given a complaint checks if it is valid according to the activation rules in line 81 or line 85, and when given a set of shares checks that they are valid and reruns the activation rule in line 89.

Remark 1 (Weak agreement vs. honest-dealer aggreement) One can also aim for an *honest-dealer agreement* property, where, if D is honest, then honest parties output the same coin value. Our *weak agreement* property is stronger: if D misbehaves, then some parties may output $\perp$, but honest parties will never output different coin values. It reduces to *honest-dealer agreement* by having parties flip a local coin whenever $\perp$ is output. □

6 Weak Multiple-Dealer Coin-Flip

In this section we define the weak multiple-dealer coin-flip (wMDCF) protocol. It is *weak* as it inherits the agreement property from wHDCF: parties may output $\perp$, but if two honest parties output a value in $\{0, 1\}$, then it will be the same. It is called *multiple-dealer* as there are multiple dealers, forming a *proposers committee*, selected pseudorandomly using $\mathsf{SampleCommittee}_p()$. The protocol uses parameters m_{wMDCF} and w_{wMDCF}. Parameter m_{wMDCF} refers to the size of the proposers committee, i.e., the number of parties that are selected to act as a dealer in an instance of wMDCF. Parameter w_{wMDCF} refers to the number of parties in the proposers committee we asynchronously wait for. In Section 7 we show how to set these parameters, such that at least one good setup appears on the ledger, except with negligible probability, and a constant rate γ of the setups are good.

Syntax. The syntax of weak Multiple-Dealer Coin-Flip (wMDCF) is as follows:

Deal On input $(\text{DEAL}, \mathsf{sid})$ a participating party starts running the dealing protocol and may as a result help produce a public output PubOutMultiDeal.

Flip On input $(\text{FLIP}, \mathsf{sid}, \mathsf{cid})$ for a coin identifier cid, after $\mathsf{PubOutMultiDeal}(\mathsf{sid}, \mathsf{Ledger}) \neq \perp$, a party starts running the coin-flip protocol and outputs $(\text{DONE-FLIP}, \mathsf{sid}, \mathsf{cid}, s)$, where $s \in \{\perp\} \cup \{0, 1\}^{\lambda}$.

Security. The security properties of honest-dealer coin-flip are as follows. For the *agreement* and *unpredictability* properties we use a probability $\gamma > 0$, called the *good-setup probability*, which depends on the parameter w_{wMDCF} and on the hiding probability β of wHDCF, and is constant and independent of sid and cid.

Termination (1)If all honest parties get input $(\textsc{deal}, \mathsf{sid})$ then eventually there is public output $\mathsf{PubOutMultiDeal}(\mathsf{sid}, \mathsf{Ledger}) \neq \perp$, except with negligible probability.

(2) If all honest parties get input $(\textsc{flip}, \mathsf{sid}, \mathsf{cid})$ then eventually all honest parties give an output $(\textsc{done-flip}, \mathsf{sid}, \mathsf{cid}, \cdot)$, except with negligible probability.

γ-Agreement For each session sid and coin identifier cid it holds that, if two honest parties output $(\textsc{done-flip}, \mathsf{sid}, \mathsf{cid}, c_{\mathsf{P}})$ and $(\textsc{done-flip}, \mathsf{sid}, \mathsf{cid}, c_{\mathsf{Q}})$, such that $c_{\mathsf{P}} \neq \perp$ and $c_{\mathsf{Q}} \neq \perp$, then $c_{\mathsf{P}} = c_{\mathsf{Q}}$, except with negligible probability. Moreover, with probability γ it holds that no honest party outputs $\perp$ as the value of the coin. All together, this means that, if two honest parties have outputs $(\textsc{done-flip}, \mathsf{sid}, \mathsf{cid}, c_{\mathsf{P}})$ and $(\textsc{done-flip}, \mathsf{sid}, \mathsf{cid}, c_{\mathsf{Q}})$, then $c_{\mathsf{P}} = c_{\mathsf{Q}} \neq \perp$ with probability γ.

γ-Unpredictability For each session sid and coin identifier cid it holds that the value of coin cid is unpredictable with probability γ.

In Appendix D we formalize the *agreement* and *unpredictability* properties, and show the proofs.

Algorithm 7 Scheme wMDCF, algorithm Deal, where an instance sid of wMDCF is created at some point p on Ledger. Code for process P_i.

```
    State:
97:     $\mathsf{setups}[w_{\mathsf{wMDCF}}] \leftarrow [\,]$

98: upon input $(\textsc{deal}, \mathsf{sid})$ do
99:     $\mathcal{C} \leftarrow \mathsf{SampleCommittee}(\mathsf{sid}, m_{\mathsf{wMDCF}})$
100:    for $j \in [m_{\mathsf{wMDCF}}]$ such that $\mathcal{C}[j] = P_i$ do
101:        $\mathsf{wHDCF}(\mathsf{Deal}, ((\mathsf{P}_i, j), \mathsf{sid}))$

102: upon $w_{\mathsf{wMDCF}}$ setups $\mathsf{PubOutMultiDeal}(((\mathsf{P}_j, k), \mathsf{sid}), \mathsf{Ledger}) \neq \perp$ where $\mathcal{C}[k] = \mathsf{P}_j$
103:    Let setups contain the identifiers which gave public output sorted deterministically
104:    $\mathsf{seed}(\textsc{seed}, \mathsf{sid})$
```

Construction. On Algorithm 7 we implement Deal. On input $(\textsc{deal}, \mathsf{sid})$, a protocol instance is created with some starting point p. For each time P_i is sampled to be a dealer in a wHDCF instance (line 99), it creates a new instance of wHDCF and runs the Deal algorithm. Every party waits for w_{wMDCF} instances of the wHDCF protocol (started by the dealers sampled in line 99) to give public output on the Ledger. When this happens, parties run an instance of the seed protocol (line 104). This seed will be later used in the Flip algorithm of wMDCF to pseudorandomly choose one of the w_{wMDCF} setups. We define $\mathsf{PubOutMultiDeal} = \mathsf{PubOutSeedOpen}$, so the output of the seed protocol signals the end of the dealing phase.

In Algorithm 8 we implement Flip. On input $(\textsc{flip}, \mathsf{sid}, \mathsf{cid})$ and after observing public output PubOutMultiDeal every party P_i uses a cryptographic hash function H, to hash $(\mathsf{sid}, \mathsf{cid}, \mathsf{seed})$ into

Algorithm 8 Scheme wMDCF, algorithm Flip (cid), where an instance sid of wMDCF is created at some point p on Ledger. Code for process P_i.

```
105: upon input (FLIP, sid, cid) such that $\mathsf{PubOutMultiDeal}(\mathsf{sid}, \mathsf{Ledger}) \neq \perp$ do
106:     $j \leftarrow H(\mathsf{sid}, \mathsf{cid}, \mathsf{PubOutMultiDeal}(\mathsf{sid}, \mathsf{Ledger}))$
107:     $\mathsf{wHDCF}(\textsc{flip}, \mathsf{setups}[j], \mathsf{cid})$

108: upon deliver $(\textsc{done-flip}, \mathsf{sid}, \mathsf{cid}, s, \pi)$
109:     if $\mathsf{wHDCFVerify}(\pi, s)$ then
110:         output $(\textsc{done-flip}, \mathsf{sid}, \mathsf{cid}, s)$
```

$j \in \{1, \ldots, w_{\mathsf{wMDCF}}\}$ (line 106). Then, the algorithm Flip of the wHDCF_j instance is used to compute the value of coin cid. We assume that each party on the committee of the selected wHDCF instance disseminate the output to the ground population.

7 Setting the Parameters

Definition 2 (Binomial distribution) Let X a random variable counting the number of successes out of n trials, where success happens with probability p. Then X follows the binomial distribution, i.e., $X \sim \mathcal{B}(n, p)$ and the probability that exactly k successes happen is

$$\Pr[X = k] = \Pr[\mathcal{B}(n, p) = k] = \binom{n}{k} p^k (1-p)^{(n-k)}. \tag{2}$$

7.1 Sampling a holding committee for wVSS and wHDCF

Let n denote the size of a holding committee and $\tau < n/2$ denote a number, such that the holding committee has at most τ corruptions with a constant probability β, and more than $n-\tau$ corruptions only with a negligible probability $\epsilon = 2^{-\lambda}$, where λ is the security parameter. The idea is the following. If we use a (n, τ)-secet-sharing or common-coin scheme in the committee, then the committee is *hiding* with probability β and *live* with probability $1-\epsilon$. These capture the parameters of both the wVSS and the wHDCF schemes. In wVSS we have $n \triangleq n_{\text{VSS}}$ and $\tau \triangleq \tau_{\text{VSS}}$, and in wHDCF we have $n \triangleq n_{\text{COIN}}$ and $\tau \triangleq \tau_{\text{COIN}}$.

As discussed earlier, we model a committee-election mechanism as a black-box function SampleCommittee(), which samples parties with probability proportional to their stake at some well-defined point on the ledger. In practice, this can be achieved by replacing each party with a (usually very large) number of smaller, atomic sub-parties, proportional to each party's stake, and use a VRF to pseudorandomly choose a sub-party [26]. In this section we assume that the ground population (the number of sub-parties) is very large, so that the probability of choosing a corrupted party does not change after choosing a party. Hence, SampleCommittee() does sampling with replacement, which can be modelled with a binomial distribution.

Using (2) we have that $\beta = \sum_{k=0}^{\tau} \Pr[\mathcal{B}(n, 1-p) = k]$ and $\epsilon = \sum_{k=n-\tau+1}^{n} \Pr[\mathcal{B}(n, 1-p) = k]$, for $p = 2/3$. In Table 1 we show various combinations for n and τ, such that $\epsilon \leq 2^{-\lambda}$ for $\lambda = 60$, and the resulting hiding probability β.

7.2 Sampling a proposer committee for seed and wMDCF

In protocols seed and wMDCF parties have a chance to participate in the *proposers committee*, i.e., to win the right to become a dealer in a wVSS or wHDCF instance, respectively. Parties are again sampled using SampleCommittee (Section 7.1), which returns a committee of size m, but the protocols only wait for the first w setups to appear on Ledger and only use those. In seed, we have $m \triangleq m_{\text{SEED}}$ and $w \triangleq w_{\text{SEED}}$, and in wMDCF we have $m \triangleq m_{\mathsf{wMDCF}}$ and $w = w_{\mathsf{wMDCF}}$.

Necessary conditions. As before, we need to make sure that, except with negligible probability $\epsilon = 2^{-\lambda}$, there are at least w honest parties on the committee to ensure termination. This is bounded as ϵ in Section 7.1 but with n and τ replaced by m and $w-1$ respectively. But now we additionally need to make sure that, except with negligible probability at least one of the w setups that appear on Ledger is a *good setup*, that is, from an honest party who sampled a committee with less than τ corruptions. This condition corresponds exactly to the setup in Section 7.1 being hiding, but with the probability p changed to account for the fact that we are interested in the probability of not just an honest party but an honest party *who provided a good setup* making it into any subset of size w. Since an honest dealer has a β (which depends on the parameters of the subprotocol) probability of providing a bad setup, we set $p = \beta \cdot 2/3$ and require $\sum_{k=0}^{w-1} \Pr[\mathcal{B}(m, 1-p) = k] \geq 1-2^{-\lambda}$.

Good-setup probability. Finally, specifically for wMDCF, we calculate the probability γ, defined in Section 6, that a setup published on Ledger is good, i.e., the probability of getting an unpredictable and agreed upon value in each coin flip. We derive this from the expected number of bad setups, which (by linearity of expectation) is $m \cdot (1-\beta \cdot \frac{2}{3})$, and from the fact that the adversary can schedule the order of messages, causing all bad setups and, hence, only $w - m \cdot (1-\beta \cdot \frac{2}{3}))$ good setups, to appear on Ledger. This gives us the fraction of good setups that in expectation appear on the ledger as

$$\gamma = \frac{w - (m \cdot (1 - \beta \cdot \frac{2}{3}))}{w}. \tag{3}$$

Putting it all together. We show the resulting parameters with $\lambda = 60$ bits of security in Table 1. As an example, for a holding committee with size $n = 259$ and reconstruction threshold $\tau = 103$, we get hiding probability $\beta = 98.7\%$. Then we can sample a proposers committee of size $m = 653$ and wait for $w = 327$. This results in $84,693$ encrypted shares being posted on the Ledger , and for wMDCF it gives a good-setup probability $\gamma = 31.8\%$.

8 Analysis of Communication Complexity

To demonstrate the power of being able to sample concretely small committees, we analyze the concrete complexity of our protocols. Note that a purely asymptotic analysis would not show any gains over simply using a state of the art ADKG protocol with subset sampling and near optimal resilience. We give all sizes in *bits*, but for simplicity we treat group and field elements as λ bits. For instance, we use 3λ as the size of an encrypted share, which (using section 2.2.1) consists of 2 group elements and a symmetrically encrypted share of a secret of size λ. This would not be precise for concrete instantiations, but it would only change our estimates by a small constant factor which depends, for example, on the concrete curves being employed.

n	τ	β	m	w	γ	$n \cdot w$
653	320	$> 1 - 2^{60}$	653	321	32.2%	209.6K
300	125	99.9%	653	322	32.3%	96.6K
280	114	99.6%	653	323	32.1%	90.4K
275	111	99.4%	653	324	32.0%	89.1K
271	109	99.3%	653	325	32.0%	88.1K
265	106	99.0%	653	326	31.9%	86.4K
261	104	98.8%	653	327	31.9%	85.3K
259	103	98.7%	653	327	31.8%	84.7K
257	102	98.6%	659	330	31.6%	84.8K
256	101	98.3%	672	337	31.3%	86.3K
254	100	98.1%	682	342	31.1%	86.9K
252	99	98.0%	692	347	30.8%	87.4K

Table 1: This table shows possible values (subject to conditions in Section 7.1) for the *holding committee* parameters, n and τ, and the resulting hiding probability β. For each obtainable β, it shows possible values (subject to conditions in Section 7.2) for the *proposers committee* parameters, m and w, and the resulting good-setup probability γ. In both seed and wMDCF schemes, each of the w dealers encrypts keys for a committee of size n, which gives a total of $m * w$ encryptions.

We define ATOB complexity as the cost of including a message of a given size in Ledger. In the following "broadcast" refers to broadcasting through the ATOB and "multicast" refers to a party sending a message to all parties in the ground population. As the communication cost of a broadcast and multicast depends on the concrete implementation we keep these costs opaque and report the results as a number of broadcasts and multicasts of various sizes. For intercommittee communication we assume point to channels are used and give the results in total number of bits sent though the channels.

The wVSS protocol has an ATOB complexity of 1 message of size $n_{\text{VSS}} \cdot 3\lambda + \lambda$ from the encrypted shares and masked message in the setup. To distribute the output then either the secret of size λ is multicast, or a complaint of size at most $(\tau_{\text{VSS}} + 1) \cdot 2\lambda$ (in case of validly encrypted shares reconstructing an inconsistent setup) is multicast and the dealer can be proven malicious. A priori every member of the committee needs to multicast the output, giving a multicast complexity of n_{VSS} messages of size λ (or of size at most $(\tau_{\text{VSS}} + 1) \cdot 2\lambda$, in which case the dealer can be slashed). The remaining interaction constists of n_{VSS} committee members sending one message with a decrypted share or complaint and proof of total size at most 4λ to the rest of the committee, resulting in a total message complexity of at most ${n_{\text{VSS}}}^2 \cdot 4\lambda$.

The seed protocol does not add any interaction besides running m_{SEED} instances of wVSS. Only the first w_{SEED} of those to make it onto the ledger will result in interaction between committe members, so the communication complexity is at most $m_{\text{SEED}} \cdot {n_{\text{VSS}}}^2 \cdot 4\lambda$. The ATOB complexity of the deal phase of seed is m_{SEED} messages of size $n_{\text{VSS}} \cdot 3\lambda + \lambda$. To disseminate the outputs there is an additional $(w_{\text{SEED}} - s) \cdot n_{\text{VSS}}$ multicasts of size λ and $s \cdot n_{\text{VSS}}$ multicasts of size at most $(\tau_{\text{VSS}} + 1) \cdot 2\lambda$, where s is the number of parties that can be slashed.

The wHDCF protocol has an ATOB complexity of 1 message of size $n_{\text{COIN}} \cdot 3\lambda + \lambda$ and no additional communication in the intial setup phase. The message complexity of each flip is at most ${n_{\text{COIN}}}^2 \cdot 4\lambda$ from reconstructing the coin (or $\perp$) in the committee, and then to dessiminate the value to the ground population each committee member multicasts the reconstructed coin or a complaint of size at most 4λ, resulting in at most ${n_{\text{COIN}}}^2 \cdot 4\lambda$ bits communicated in addition to n_{COIN} multicasts of size 4λ.

The deal phase of the wMDCF protocol has the same complexity as m_{wMDCF} deal phases of wHDCF and a single run of the seed protocol. I.e. an ATOB complexity of m_{wMDCF} messages of size $n_{\text{COIN}} \cdot 3\lambda + \lambda$ and m_{SEED} messages of size $n_{\text{VSS}} \cdot 3\lambda + \lambda$, a multicast complexity of $w_{\text{SEED}} \cdot n_{\text{VSS}}$ multicasts of size at most $(\tau_{\text{VSS}}+1)\cdot 2\lambda$, in addition to a communication complexity of $m_{\text{SEED}} \cdot {n_{\text{VSS}}}^2 \cdot 4\lambda$ bits. Whenever a coin needs to be flipped using wMDCF, the message complexity is that of running the selected wHDCF protcol.

To refresh the setup after the stake distribution has changed, one would need to first run an instance of the seed protocol and then the deal phase of the wMDCF protocol. Using the best parameters in Table 1 the concrete cost of refreshing the setup is 1959 messages of size 778λ, and 169386 multicasts of size at most 208λ. The communication complexity of flipping a coin and disseminating it to all parties is $259^2 \cdot 4\lambda$ in addition to 259 multicasts of 4λ bits. Employing the optimizations in Remark 2 reduces the multicast complexity of refreshing the setup to 654 messages of size at most $(\tau_{\text{VSS}} + 1) \cdot 2\lambda$. Similarly the cost of distributing a coin becomes the same as 1 multicast of size 4λ.

If we were to assume $t < 0.3n$ in the paradigm of “subset sampling with almost optimal resilience” [4], and need a committee with honest supermajority with probability $1 - 2^{-60}$, then one would need to sample a committee with 16037 parties [22]. If we then instantiate a state of the art ADKG protocol with $O(n^3\lambda)$ communication using the committee and for the sake of an example assume the concrete cost is $n^3\lambda$, then we get a complexity of $> 4 \cdot 10^{12}\lambda$. It is then clear that our approach is far cheaper for all but extremely large values of n.

Remark 2 (Deduplicating multicasts) Notice that each party only needs to receive a single proof of output for each of the w_{SEED} wVSS setups. Since large scale P2P networks usually employ gossiping with deduplication of previously forwarded messages, each node can consider different output justifications from the same wVSS “identical” for the purpose of decuplication. We conjecture that in most gossip based P2P networks this results in a communication cost which is less than that of a single multicast, as it can be seen as a multicast from a single source which has gotten a headstart by being predistributed to $O(\lambda)$ nodes. With this instantiation the cost of disseminating the wVSS outputs from the seed protocol becomes the same as multicasting w_{SEED} messages of size at most $(\tau_{\text{VSS}} + 1) \cdot 2\lambda$ over the gossip network. The same deuplication trick can be employed when disseminating the coin flips, reducing the cost of distributing the coin to the same as 1 multicast of size 4λ. □

9 Asynchronous Total-Order Broadcast with Weak Coins

It is known that coins consumed by Byzantine agreement protocols do not need to be perfect, i.e., unpredictable and agreed upon with all but negligible probability [11, 32]. However, many many existing ATOB protocols [27, 18, 39] base their correctness on a global perfect coin. While one can add machinery on top of our weak coin wMDCF to instantiate a perfect coin, in this section we show how to modify a concrete protocol, namely DAG-rider [27], to use our weak coin functionality. We use f to denote the number of corrupt nodes to stay consistent with their notation. The modification simply adds an extra round at the end of each wave in which each party adds its observed coin output their vertex and changes the consensus logic to depend on these reported coins instead of a common coin.

To keep the presentation concise, we present the changes relative to the algorithms and line numbers from the original paper [27]. Specifically, we modify Algorithms 1, 2, and 3 [27]. For Algorithm 3 [27], which contains the core consensus logic, we also present its modified version in Algorithm 9.

First, we need the output of the common-coin protocol in the vertices in the last round of each wave, so we add the field coin to the vertex structure, as shown in Algorithm 1 of DAG-Rider [27]. The field contains a leader or a proof that wMDCF has output $\perp$. In rounds that are not congruent to $4 \mod 5$ the field is left empty. We use these coin values to commit the leader of the wave, which now consists of five rounds, hence we change the "4" in lines 11 and 12 of Algorithm 2 [27] to "5". Additionally, we add the coin value to the new vertices after line 20 by checking if "$r \mod 5 = 4$" and then setting "$v.\mathsf{coin} \leftarrow \mathsf{chooseLeader}(r/5)$", and we strengthen the condition in line 25 to also verify that vertices in the last round of each wave include a valid coin or a justified output of $\perp$.

Finally, we modify the consensus logic in Algorithm 3 [27], so that get_wave_vertex_leader uses the coins included in the DAG to potentially pick a leader, and the wave_ready activation rule commits a leader when more than f nodes in a wave report the coin output (in addition to the existing condition for picking a leader from a common core of $2f+1$ vertices). Similarly, the check in line 41 of Algorithm 3 [27] is strengthened, requiring additionally that the coin that picks the leader is included in the DAG of the newly committed leader through a strong path. These changes are described in detail in Algorithm 9.

Algorithm 9 DAG-rider with weak coins.

```
111:upon wave_ready($w$) do
112:    $v \leftarrow$ get_wave_vertex_leader($w$)
113:    if $v = \perp \vee |\{v' \in \mathsf{DAG}_i[\mathsf{round}(w,5)] : v'.\mathsf{coin} = v.\mathsf{source}\}| < f+1$
114:            $\vee |\{v' \in \mathsf{DAG}_i[\mathsf{round}(w,4)] : \mathsf{strong\_path}(v',v)\}| < 2f+1$ then
115:        return
116:    leadersStack.push($v$)
117:    for wave $w'$ from $w-1$ down to decidedWave $+ 1$ do
118:        $v' \leftarrow$ get_wave_vertex_leader($w'$)
119:        if $v' \neq \perp \wedge \mathsf{strong\_path}(v,v') \wedge \exists v'' \in \mathsf{DAG}_i[\mathsf{round}(w',5)]$ :
120:            $v''.\mathsf{coin} = v.\mathsf{source} \wedge \mathsf{strong\_path}(v,v'')$ then
121:            leadersStack.push($v'$)
122:            $v \leftarrow v'$
123:        decidedWave $\leftarrow w$
124:        order_vertices(leadersStack)

125:procedure get_wave_vertex_leader($w$)
126:    if $\exists v \in \mathsf{DAG}_i[\mathsf{round}(w,1)], v' \in \mathsf{DAG}_i[\mathsf{round}(w,5)] : v.\mathsf{source} = v'.\mathsf{coin}$ then
127:        return $v$
128:    return $\perp$
```

Consistency. If a wave leader is committed in some view of the DAG in wave w, then at least $f+1$ of the vertices in round 5 of w reported a coin output. Assume in some in wave $w' \geq w$ a wave vertex leader is committed in some other view of the DAG. If $w = w'$ then leaders and vertices are consistent by the agreement property of the coin and vertices being sent through reliable broadcast. If $w < w'$ then the leader in wave w' is connected to $n - f$ vertices in round 5 of w through a strong path. Since the vertices are reliably broadcast, at least one of these overlap with the at least $f + 1$ vertices reporting a coin output, which allowed committing a leader in W. Hence, the leader of round w will be pushed to leadersStack in line 121, ensuring consistency.

Liveness. Everything is as before, but there are now more ways in which get wave vertex leader could fail to return a vertex. Before it could only happen when the leader is not in the common core, which has size $2n/3$ and was committed before the coin value was known. Thus, the wave vertex leader has output with prob. $2/3$. In our version there is some constant probability that the coin is bad (meaning that there could be disagreement or the coin could have been predicted). The adversary can schedule the waves in which the coin is bad such that no leader can be committed. If the coin is good, then all vertices in round 5 will report the output, the vertices in the common core are independent of the coin value. So the probability of committing a wave is $\Pr[\mathit{good}] \cdot \frac{2}{3}$.

10 Conclusion

In this work we have presented protocols for generating randomness in an asynchronous PoS setting with dynamic participation. The protocols are practical and concretely efficient, they employ no trusted setup, and they make use of small committees. We have computed concrete numbers for the committee size. Specifically, we can have a committee of $m = 653$ proposers, each generating a setup for $n = 359$ holders, resulting in approx. $85K$ encrypted values posted on Ledger. For $\kappa = 60$ bit of security and assuming optimal corruption $1/3$ in the ground population, this gives randomness-generation protocols that are live with all but negligible probability. Our common-coin protocol is unpredictable and agreed-upon with probability approx. 31.8%, and, as it is based on threshold cryptography, the setup can be used for a flipping a polynomial number of coins. These committee sizes result from the fact that we require not all but only a constant factor of our setups to be good.

It is instructive to compare these results against previous literature, particularly against the approach that runs the randomness-generation protocols in committees with honest supermajority. Algorand [26, Figure 3] requires a committee of size approx. 2000, assuming corruption 0.2 in the ground population, and larger than 4000, assuming corruption 0.24, to get good committees with probability $5 \cdot 10^{-9}$, or approx. 28 bits of security. Extending this approach to a ground population with corruption 0.3, which is still sub-optimal, and 60 bits of security, the authors of GearBox [22] show that committees of size 16037 are needed. We remark that asynchronous distributed key generation protocols, the state-of-the-art approach for threshold-setup generation, require honest supermajority, hence one would require a committee of similar sizes and sub-optimal resilience in the ground population.

Acknowledgments

This work has received funding from the Swiss National Science Foundation (SNSF) under grant agreement Nr. 200021_188443 (Advanced Consensus Protocols). Simon Holmgaard Kamp and Jesper Buus Nielsen are partially funded by The Concordium Foundation.

References

[1] I. Abraham, P. Jovanovic, M. Maller, S. Meiklejohn, and G. Stern. Bingo: Adaptively secure packed asynchronous verifiable secret sharing and asynchronous distributed key generation. *IACR Cryptol. ePrint Arch.*, page 1759, 2022.

[2] I. Abraham, P. Jovanovic, M. Maller, S. Meiklejohn, G. Stern, and A. Tomescu. Reaching consensus for asynchronous distributed key generation. In *PODC*, pages 363–373. ACM, 2021.

[3] M. Ben-Or. Another advantage of free choice: Completely asynchronous agreement protocols (extended abstract). In *PODC*, pages 27–30. ACM, 1983.

[4] E. Blum, J. Katz, C. Liu-Zhang, and J. Loss. Asynchronous byzantine agreement with subquadratic communication. In *TCC (1)*, volume 12550 of *Lecture Notes in Computer Science*, pages 353–380. Springer, 2020.

[5] A. Boldyreva. Threshold signatures, multisignatures and blind signatures based on the gap-diffie-hellman-group signature scheme. In *Public Key Cryptography*, volume 2567 of *Lecture Notes in Computer Science*, pages 31–46. Springer, 2003.

[6] D. Boneh, J. Bonneau, B. Bünz, and B. Fisch. Verifiable delay functions. In *CRYPTO (1)*, volume 10991 of *Lecture Notes in Computer Science*, pages 757–788. Springer, 2018.

[7] D. Boneh, B. Lynn, and H. Shacham. Short signatures from the weil pairing. *J. Cryptol.*, 17(4):297–319, 2004.

[8] C. Cachin, K. Kursawe, F. Petzold, and V. Shoup. Secure and efficient asynchronous broadcast protocols. In *CRYPTO*, volume 2139 of *Lecture Notes in Computer Science*, pages 524–541. Springer, 2001.

[9] C. Cachin, K. Kursawe, and V. Shoup. Random oracles in constantinople: Practical asynchronous byzantine agreement using cryptography. *J. Cryptol.*, 18(3):219–246, 2005.

[10] J. Camenisch, M. Drijvers, T. Hanke, Y. Pignolet, V. Shoup, and D. Williams. Internet computer consensus. In *PODC*, pages 81–91. ACM, 2022.

[11] R. Canetti and T. Rabin. Fast asynchronous byzantine agreement with optimal resilience. In *STOC*, pages 42–51. ACM, 1993.

[12] I. Cascudo and B. David. SCRAPE: scalable randomness attested by public entities. In *ACNS*, volume 10355 of *Lecture Notes in Computer Science*, pages 537–556. Springer, 2017.

[13] I. Cascudo and B. David. ALBATROSS: publicly attestable batched randomness based on secret sharing. In *ASIACRYPT (3)*, volume 12493 of *Lecture Notes in Computer Science*, pages 311–341. Springer, 2020.

[14] J. Chen, S. Gorbunov, S. Micali, and G. Vlachos. ALGORAND AGREEMENT: super fast and partition resilient byzantine agreement. *IACR Cryptol. ePrint Arch.*, page 377, 2018.

[15] K. Choi, A. Arun, N. Tyagi, and J. Bonneau. Bicorn: An optimistically efficient distributed randomness beacon. *IACR Cryptol. ePrint Arch.*, page 221, 2023.

[16] K. Choi, A. Manoj, and J. Bonneau. Sok: Distributed randomness beacons. *IACR Cryptol. ePrint Arch.*, page 728, 2023.

[17] S. Cohen, I. Keidar, and A. Spiegelman. Not a coincidence: Sub-quadratic asynchronous byzantine agreement WHP. In *DISC*, volume 179 of *LIPIcs*, pages 25:1–25:17. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020.

[18] G. Danezis, L. Kokoris-Kogias, A. Sonnino, and A. Spiegelman. Narwhal and tusk: a dag-based mempool and efficient BFT consensus. In *EuroSys*, pages 34–50. ACM, 2022.

[19] S. Das, V. Krishnan, I. M. Isaac, and L. Ren. Spurt: Scalable distributed randomness beacon with transparent setup. In *IEEE Symposium on Security and Privacy*, pages 2502–2517. IEEE, 2022.

[20] S. Das, T. Yurek, Z. Xiang, A. Miller, L. Kokoris-Kogias, and L. Ren. Practical asynchronous distributed key generation. In *IEEE Symposium on Security and Privacy*, pages 2518–2534. IEEE, 2022.

[21] B. David, P. Gazi, A. Kiayias, and A. Russell. Ouroboros praos: An adaptively-secure, semi-synchronous proof-of-stake blockchain. In *EUROCRYPT (2)*, volume 10821 of *Lecture Notes in Computer Science*, pages 66–98. Springer, 2018.

[22] B. David, B. Magri, C. Matt, J. B. Nielsen, and D. Tschudi. Gearbox: Optimal-size shard committees by leveraging the safety-liveness dichotomy. In *CCS*, pages 683–696. ACM, 2022.

[23] Drand. A distributed randomness beacon daemon, 2022. `https://drand.love`.

[24] P. Feldman. A practical scheme for non-interactive verifiable secret sharing. In *FOCS*, pages 427–437. IEEE Computer Society, 1987.

[25] M. J. Fischer, N. A. Lynch, and M. Paterson. Impossibility of distributed consensus with one faulty process. In *PODS*, pages 1–7. ACM, 1983.

[26] Y. Gilad, R. Hemo, S. Micali, G. Vlachos, and N. Zeldovich. Algorand: Scaling byzantine agreements for cryptocurrencies. In *SOSP*, pages 51–68. ACM, 2017.

[27] I. Keidar, E. Kokoris-Kogias, O. Naor, and A. Spiegelman. All you need is DAG. In *PODC*, pages 165–175. ACM, 2021.

[28] V. King and J. Saia. Byzantine agreement in expected polynomial time. *J. ACM*, 63(2):13:1–13:21, 2016.

[29] V. King and J. Saia. Correction to byzantine agreement in expected polynomial time, JACM 2016. *CoRR*, abs/1812.10169, 2018.

[30] A. K. Lenstra and B. Wesolowski. A random zoo: sloth, unicorn, and trx. *IACR Cryptol. ePrint Arch.*, page 366, 2015.

[31] S. Micali, M. O. Rabin, and S. P. Vadhan. Verifiable random functions. In *FOCS*, pages 120–130. IEEE Computer Society, 1999.

[32] A. Patra, A. Choudhury, and C. P. Rangan. Asynchronous byzantine agreement with optimal resilience. *Distributed Comput.*, 27(2):111–146, 2014.

[33] Protocol Labs. Filecoin: A decentralized storage network. `https://filecoin.io/filecoin.pdf`, 2017.

[34] M. O. Rabin. Randomized byzantine generals. In *FOCS*, pages 403–409. IEEE Computer Society, 1983.

[35] M. Raikwar and D. Gligoroski. Sok: Decentralized randomness beacon protocols. In *ACISP*, volume 13494 of *Lecture Notes in Computer Science*, pages 420–446. Springer, 2022.

[36] D. A. W. Ronald L. Rivest, Adi Shamir. Time-lock puzzles and timed-release crypto. Technical report, 1996.

[37] P. Schindler, A. Judmayer, N. Stifter, and E. R. Weippl. Hydrand: Efficient continuous distributed randomness. In *IEEE Symposium on Security and Privacy*, pages 73–89. IEEE, 2020.

[38] A. Shamir. How to share a secret. *Commun. ACM*, 22(11):612–613, 1979.

[39] A. Spiegelman, N. Giridharan, A. Sonnino, and L. Kokoris-Kogias. Bullshark: DAG BFT protocols made practical. In *CCS*, pages 2705–2718. ACM, 2022.

[40] E. Syta, P. Jovanovic, E. Kokoris-Kogias, N. Gailly, L. Gasser, I. Khoffi, M. J. Fischer, and B. Ford. Scalable bias-resistant distributed randomness. In *IEEE Symposium on Security and Privacy*, pages 444–460. IEEE Computer Society, 2017.

A Proofs for wVSS

In this section we prove the following.

Theorem 1 *The scheme* wVSS *given in algorihm 1 and algorihm 2 is a Weak VSS as defined in section 3.* □

We first formalize the notion of weak hiding.

Definition 3 (β-Weak Hiding) The following two properties should hold:

β-Sometimes Good: For all sid there should exist an event $\mathsf{Bad}_{\mathsf{sid}}$ which can be computed in PPT from the view of the adversary once the first honest party gave public output PubOutVSSCommit for sid, i.e., the adversary knows if the event happened. It should hold that if D is honest in sid then $\mathsf{Bad}_{\mathsf{sid}}$ happens with probability at most $1 - \beta$, independently for each sid.

Hiding when Good: Furthermore, the adversary can win the following game with probability at most negligibly better than $1/2$. The adversary may initiate as many sessions sid as it wants. At any point it may specify a challenge $(\text{COMMIT}, \mathsf{sid}, m_0, m_1)$ for a sid for which no commitment was made yet. Then the game flips a uniformly random bit b_{sid} and gives input $(\text{COMMIT}, \mathsf{sid}, m_{b_{\mathsf{sid}}})$ to the dealer D specified by sid. The adversary may challenge several sessions. The adversary wins the game if at some point it outputs (sid, g) where sid is a session with an honest dealer, where a challenge was made, where an honest party has public output $\mathsf{PubOutVSSCommit}$, where $\mathsf{Bad}_{\mathsf{sid}}$ did not happen, where the first honest party still did not get input OPEN, and $g = b_{\mathsf{sid}}$. □

PROOF theorem 1 We will assume that the committee selection for wVSS is done such that with constant probability at least β there are at most τ_{VSS} corrupted parties and such that except with negligible probability there are less than $n_{\text{VSS}} - \tau_{\text{VSS}}$ corruption except with negligible probability. We have to prove **Termination (1)–(3)**, **Binding**, **Validity**, **β-Sometimes Good**, and **Hiding when Good**.

Termination (1). Assume that D is honest and gets input (COMMIT, m), and all other honest parties get input $(\text{COMMIT}, \mathsf{sid})$. D can perform all the operations locally and broadcast $(\mathsf{sid}, \pi, (e_1, \mathsf{H}_1) \ldots, (e_{n_{\text{VSS}}}, \mathsf{H}_{n_{\text{VSS}}}), m_{\text{MASKED}})$, which by definition of $\mathsf{PubOutVSSCommit}$ leads to the require public output.

Termination (2). Assume $\mathsf{PubOutVSSCommit}$ occurred and all honest parties get input $(\text{OPEN}, \mathsf{sid})$. We have to show that eventually all honest parties give output DONE-OPEN. Since for all $\mathsf{P} in \mathsf{Honest}$ $((e_1, \mathsf{H}_1) \ldots, (e_{n_{\text{VSS}}}, \mathsf{H}_{n_{\text{VSS}}}), m_{masked}) = \mathsf{PubOutVSSCommit}(\mathsf{Ledger}_{\mathsf{P}}, \mathsf{sid})$. At this point either all honest ciphertexts e_i are valid or some honest ciphertext is invalid. We do a proof by case. Let us first assume that some honest ciphertext is invalid. In that case some honest party will eventually send a COMPLAINTENCRYPTION-message which will eventually arrive at all other honest parties and make them output $(\text{DONE-OPEN}, \mathsf{sid}, \perp)$ if they did not already give a DONE-OPEN-output. Assume then that all honest ciphertexts are valid. In that case all honest parties will send (SHARE, s_e, r_e). There is less than $n_{\text{VSS}} - \tau_{\text{VSS}}$ corruptions except with negligible, so there is at least τ_{VSS} honest parties. Therefore all honest parties will eventually have $|\mathsf{validShares}[\mathsf{sid}]| = \tau_{\text{VSS}} + 1$ and reconstruct some σ'. At this point they will output $(\text{DONE-OPEN}, \cdot)$.

Termination (3). Assume some honest party P gives an output $(\text{DONE-OPEN}, \cdot)$. It is easy to verify that for each condition that could make this happen, the information that triggered the event at P_i will eventually propagate to all honest parties and also have them output $(\text{DONE-OPEN}, \cdot)$ (if they did not already do so).

Binding. For a given session identifier sid assume that $\mathsf{PubOutVSSCommit}(\mathsf{sid})$ occurred and therefore $(\mathsf{sid}, \pi, (e_1, \mathsf{H}_1) \ldots, (e_{n_{\text{VSS}}}, \mathsf{H}_{n_{\text{VSS}}}), m_{\text{MASKED}})$ appeared on the ledger. We show that there exist a unique m that sid can open to. We first make some helper definitions.

We say that a set of shares $\mathsf{validShares}[\mathsf{sid}]$ opens sid to m if $|\mathsf{validShares}[\mathsf{sid}]| = \tau_{\text{VSS}} + 1$ and $e_j = \mathsf{Enc}_{\mathsf{ek}_j}(s_j, r_j)$ for all $(\text{SHARE}, s_j, r_j) \in \mathsf{validShares}[\mathsf{sid}]$, and $\sigma' = \mathsf{TSS.Reconstruct}(\{s_j\}_{j \in [\tau_{\text{VSS}}+1]})$ verifies under vk_{D}, and $((e_1, \ldots, e_{n_{\text{VSS}}}), m_{\text{MASK}}) = \mathsf{commit\_value}(\sigma')$, where $(e_1, \ldots, e_{n_{\text{VSS}}})$ is the values from $(\mathsf{sid}, \pi, (e_1, \mathsf{H}_1) \ldots, (e_{n_{\text{VSS}}}, \mathsf{H}_{n_{\text{VSS}}}), m_{\text{MASKED}})$ on Ledger, and $m = m_{\text{MASKED}} \oplus m_{\text{MASK}}$.

We say that sid is *weakly openable* if there exists at least one set $\mathsf{validShares}[\mathsf{sid}]$ which opens sid to some m. We say that sid is *strongly openable* if there exists m such that for all $\mathsf{validShares}[\mathsf{sid}]$, where $|\mathsf{validShares}[\mathsf{sid}]| = \tau_{\text{VSS}} + 1$ and $e_j = \mathsf{Enc}_{\mathsf{ek}_j}(s_j, r_j)$ for all $(\text{SHARE}, s_j, r_j) \in \mathsf{validShares}[\mathsf{sid}]$, it holds that $\mathsf{validShares}[\mathsf{sid}]$ opens sid to m.

Lemma 1 sid *is weakly openable iff strongly openable.* $\square$

PROOF Obviously if sid is strongly openable it is clearly weakly openable, so it is sufficient to prove that weakly openable implies strongly openable. Assume that sid is weakly openable. Thus there exists $\sigma' = \mathsf{TSS.Reconstruct}(\{s_j\}_{j \in [\tau_{\text{VSS}}+1]})$ which verifies under $\mathsf{vk_D}$, and $((e_1, \ldots, e_{n_{\text{VSS}}}), m_{\text{MASK}}) = \mathsf{commit\_value}(\sigma')$, where $(e_1, \ldots, e_{n_{\text{VSS}}})$ is the values from $(\mathsf{sid}, \pi, (e_1, \mathsf{H}_1) \ldots, (e_{n_{\text{VSS}}}, \mathsf{H}_{n_{\text{VSS}}}), m_{\text{MASKED}})$, and $m = m_{\text{MASKED}} \oplus m_{\text{MASK}}$. Since signatures are unique it follows that $\sigma' = \mathsf{Sign}_{\mathsf{sk_D}}(\mathsf{sid})$. So, if we define $\sigma = \mathsf{Sign}_{\mathsf{sk_D}}(\mathsf{sid})$, then the value $(\mathsf{sid}, \pi, (e_1, \mathsf{H}_1) \ldots, (e_{n_{\text{VSS}}}, \mathsf{H}_{n_{\text{VSS}}}), m_{\text{MASKED}})$ on the ledger was constructed using $((e_1, \ldots, e_{n_{\text{VSS}}}), m_{\text{MASK}}) = \mathsf{commit\_value}(\sigma)$. This implies, by construction, that for *any* set $\mathsf{validShares[sid]}$, where $|\mathsf{validShares[sid]}| = \tau_{\text{VSS}} + 1$ and $e_j = \mathsf{Enc}_{\mathsf{ek}_j}(s_j, r_j)$ the shares s_j will reconstruct to σ. Since m_{MASK} is deterministically derived from σ and m_{MASKED} is fixed on Ledger it follows that $m_{\text{MASKED}} \oplus m_{\text{MASK}}$ does not depend on which $\mathsf{validShares[sid]}$ is used and hence sid is strongly openable. $\blacksquare$

Now assume that $(\mathsf{sid}, \pi, (e_1, \mathsf{H}_1) \ldots, (e_{n_{\text{VSS}}}, \mathsf{H}_{n_{\text{VSS}}}), m_{\text{MASKED}})$ appeared on Ledger. We define the unique m for which an honest party can output $(\text{DONE-OPEN}, \mathsf{sid}, m, \pi)$. Since there is less than $n_{\text{VSS}} - \tau_{\text{VSS}}$ corruptions there are at least $\tau_{\text{VSS}} + 1$ honest parties. If the encryption of any of these parties is incorrect then let $m = \perp$. Otherwise from the shares of these honest parties we get a set of shares $|\mathsf{validShares[sid]}| = \tau_{\text{VSS}} + 1$ and $e_j = \mathsf{Enc}_{\mathsf{ek}_j}(s_j, r_j)$ for all $(\text{SHARE}, s_j, r_j) \in \mathsf{validShares[sid]}$. If $\mathsf{validShares[sid]}$ opens to m' then let $m = m'$, otherwise let $m = \perp$. We argue that this m will do the job using a case proof. Assume that sid is openable. In that case it is strongly openable to some m and hence $\mathsf{validShares[sid]}$ opens to $m' = m$. In addition $(\mathsf{sid}, \pi, (e_1, \mathsf{H}_1) \ldots, (e_{n_{\text{VSS}}}, \mathsf{H}_{n_{\text{VSS}}}), m_{\text{MASKED}})$ is correctly generated from $\sigma = \mathsf{Sign}_{\mathsf{sk_D}}(\mathsf{sid})$ and therefore no honest party can be made to output $(\text{DONE-OPEN}, \mathsf{sid}, \perp)$. Assume then that sid is not openable. There is at least $\tau_{\text{VSS}}+1$ honest parties, each P_i will eventually collect $\mathsf{validShares[sid]}$ such that $|\mathsf{validShares[sid]}| = \tau_{\text{VSS}} + 1$. Since sid is not openable this set $\mathsf{validShares[sid]}$ will lead P_i to output $(\text{DONE-OPEN}, \mathsf{sid}, \perp)$, and in this case $m = \perp$

Validity. Validity follows as a special case of binding. When D is honest then clearly sid is openable to m_{D} and therefore uniquely live to m_{D}.

β-Sometimes Good. We define the event $\mathsf{Bad_{sid}}$ to be that more than τ_{VSS} parties are corrupted. Then by the assumption the event happens with probability at most $1-\beta$. Less than τ_{VSS} corrupted parties with probability τ_{VSS}.

Hiding when Good. Assume then that $\mathsf{Bad_{sid}}$ does not happen, i.e., there are at most τ_{VSS} corrupted parties. We argue that no PPT adversary $\mathcal{A}$ can make a guess g such that $g = b$ with probability p where the advantage $a = p - 1/2$ is non-negligible. For the sake of contradiction assume we have $\mathcal{A}$ contradicting this. From $\mathcal{A}$ we can construct $\mathcal{A}'$ that can make a guess g such that $g = b$ with probability p' where the advantage $a' = p' - 1/2$ is non-negligible, and which before the attack outputs an index I and then attacks the sid of the I'the instance that it creates, i.e., it tells us ahead of time which session it will attack.

We can construct $\mathcal{A}'$ as follows. Let m be a polynomial upper bound on how many instances $\mathcal{A}$ created. Let I be a uniformly random number between 1 and m and output I. Then run $\mathcal{A}$. If $\mathcal{A}$ happens to attack instance I, then run the attack of $\mathcal{A}$ and output the same g. Otherwise, if $\mathcal{A}$ starts an attack on $I' < I$ or did not do an attack by instance I, then stop running $\mathcal{A}$, attack instance I with $(m_0 = 0, m_1 = 1)$ and do a uniformly random guess g. Let $\mathcal{A}'$ be a random variable denoting the output g of $\mathcal{A}'$, let $\mathcal{A}$ denote the output g of $\mathcal{A}$, let A be the index of the session that $\mathcal{A}$ actually attacks, and let I be the guess of $\mathcal{A}'$. Note that $\Pr[I = A] = \frac{1}{m}$.

By the law of total probability we have that

$$\begin{aligned}\Pr[\mathcal{A}' = b] &= \Pr[\mathcal{A}' = b \wedge A = I] + \Pr[\mathcal{A}' = b | A \neq I] \Pr[A \neq I] \\ &= \Pr[\mathcal{A} = b \wedge A = I] + \frac{1}{2}\frac{m-1}{m} \ .\end{aligned}$$

Using again the law of total probability we get that

$$\begin{aligned}\Pr[\mathcal{A} = b \wedge A = I] &= \sum_{J=1}^{m} \Pr[\mathcal{A} = b \wedge A = I | I = J] \Pr[I = J] \\ &= \sum_{J=1}^{m} \Pr[\mathcal{A} = b \wedge A = J] \frac{1}{m} \\ &= \frac{1}{m} \sum_{J=1}^{m} \Pr[\mathcal{A} = b \wedge A = J] \\ &= \frac{1}{m} \Pr[\mathcal{A} = b] = \frac{p}{m} \\ &= \frac{p}{m} - \frac{\frac{1}{2}}{m} + \frac{1}{2}\frac{1}{m} \\ &= \frac{p - \frac{1}{2}}{m} + \frac{1}{2}\frac{1}{m} = \frac{a}{m} + \frac{1}{2}\frac{1}{m} \ .\end{aligned}$$

It follows that

$$\begin{aligned}a' = \Pr[\mathcal{A}' = b] - \frac{1}{2} &= \Pr[\mathcal{A} = b \wedge A = I] + \frac{1}{2}\frac{m-1}{m} - \frac{1}{2} \\ &= \frac{a}{m} + \frac{1}{2}\frac{1}{m} + \frac{1}{2}\frac{m-1}{m} - \frac{1}{2} \\ &= \frac{a}{m} \ .\end{aligned}$$

Since m is polynomial and a is non-negligible, it follows that a' is non-negligible.

Consider then $\mathcal{A}'$ attacking sid where we that know sid will be attacked. We will prove using a hybrids argument that $g = b$ with probability $\frac{1}{2} + \text{negl}$, reaching a contradiction. We can assume by the rules of the game that D is honest. Since $\mathsf{Bad}_{\mathsf{sid}}$ did not happen we know that at most τ_{VSS} parties of the holding committee are corrupted. Note that we will never have to run the opening phase of sid as $\mathcal{A}'$ attacks sid and therefore must make its guess g before the opening phase of sid. We now do the hybrid argument, appealing to the security of one primitive in each step, using some of them twice.

Programming the random oracle. We first show how to construct the view of the commitment phase of sid in a way computationally indistinguishable to $\mathcal{A}'$ and without using σ, and then we show how to use this in the proof. We simulate the oracle H as part of the proof, defining it lazily as it is being queried. We pick a uniformly random τ in the co-domain of H, and then we simply pretend that $\tau = H(\sigma)$, which we can do without knowing σ. If H is ever queried by the adversary on input σ such that $\mathsf{Ver}_{\mathsf{vk}_\mathsf{D}}(\sigma) = \top$, then we simply return $H(\sigma) = \tau$. This will look perfectly like the real random oracle to any adversary. In both cases $H(\sigma)$ outputs a uniformly random τ. In one case we simply sampled it before H was queried on σ.

Using a dummy signature in secret sharing. We then replace the use of σ in $\mathsf{TSS.Share}(\sigma)$ and instead share a dummy $\sigma' = 0$. This might change the distribution of the guess g of $\mathcal{A}'$ such that the advantage is no longer a'. Let $a'' = \Pr[g = b] - \frac{1}{2}$ be the advantage after the change. To show that a' is negligible it is sufficient to show that a'' is negligible and that a'' and a' are negligibly close. We first show that a'' is negligible.

Using the unforgeability of Sign. Now that we program H and use a dummy σ' we no longer use σ in the session sid. Now note that if $\mathcal{A}'$ queries H on $\sigma = \mathsf{Sign}_{\mathsf{sk}_\mathsf{D}}(\mathsf{sid})$, then it broke the signature scheme as it computed $\mathsf{Sign}_{\mathsf{sk}_\mathsf{D}}(\mathsf{sid})$ without having been given this signature: we ran the simulation without σ up until $\mathcal{A}'$ gave it to us. So, we can assume that $\mathcal{A}'$ does not query H on σ until it makes its guess g. If H is queried on σ we will stop the attack: we stop running $\mathcal{A}'$ and make a uniform guess. This will only change the advantage negligibly. We will keep calling the advantage a''. We ought to rename the new advantage a''' and keep track that a''' and a'' are negligibly close, but to avoid too many variables we will use a'' to denote the advantage after a changes when the change makes a negligible difference in the advantage.

Using the pseudorandomness of PRG. We can now assume that H is not queried on σ when we run $\mathcal{A}'$. This means that until the opening phase τ will be uniformly random in the view of the adversary. We can therefore, by security of PRG, pick ρ as a uniformly random string as opposed to $\rho = \mathsf{PRG}(\tau)$. If this changes the probability that $g = b$ noticable, then we can use this to break pseudorandomness of PRG. So we can assume that after this change the advantage a'' is still non-negligible. Now that ρ is uniform the schemes Enc and TSS are computed with uniformly random randomness, so we can appeal to their security. We can in particular use that m_{MASK} now is uniformly random (sampled from the uniform ρ) and independent of the view of the adversary. Therefore $\Pr[g = b] = \frac{1}{2}$ and we have that $a'' = 0$. We then proceed to show that a'' is close to the advantage a' of $\mathcal{A}'$ in the real game.

Using the IND-CPA of Enc. Now that ρ is uniform we can encrypt a dummy share $s_i' = 0$ in e_i for an honest P_i instead of the real share s_i. By IND-CPA security of Enc, this will not change the advantage a'' noticably as e_i is not opened until the opening phase and therefore $\mathcal{A}'$ makes its guess before we have to open e_i. We can therefore get e_i from the IND-CPA game and embed it in the execution of sid. We can of course not finish the execution of the opening phase this way, but this does not matter. Once $\mathcal{A}'$ realizes that we cannot open e_i it already gave us g. Using another hybrid argument we can do this for all honest e_i.

Using hiding of TSS. We now use that $\mathsf{Bad}_{\mathsf{sid}}$ did not happen. Notice that the analysis of the probability that $\mathsf{Bad}_{\mathsf{sid}}$ is negligible does not dependent on the protocol wVSS, it it an independent combinatorial analysis. Therefore the probability that $\mathsf{Bad}_{\mathsf{sid}}$ is still negligible after the changes in the above hybrids. This ensures that there are at most τ_{VSS} corrupted parties. So the τ_{VSS} shares in the corrupted e_i have a distribution independent of the secret σ'. We can therefore instead secret share σ. Note that before this change the probability q that $\mathcal{A}'$ queried H on σ was negligible, so if after the change the probability q' that $\mathcal{A}'$ queries H on σ is non-negligible we broke hiding of TSS: we can use the event whether H was queried on σ to guess whether σ or σ' were secret shared. Note that we secret share σ but still pick ρ uniformly at random, so we can indeed appeal to hiding of TSS at this point.

Reversing. We can then reverse the changes. First we encrypt the real shares s_i instead of dummy shares s'_i. The advantage a'' changes at most negligibly, or we could contradict IND-CPA. The same holds for the probability q that H is queried on σ. Then we replace the uniform ρ with $\rho = \mathsf{PRG}(\tau)$. The advantage a'' changes at most negligibly, or we could break pseudorandomness. The same holds for the probability q that H is queried on σ. This step is subtle as we use the seed τ in programming $H(\sigma) = \tau$ and when we do a reduction to pseudorandomness of PRG we do not get τ. Note, however, that when ρ is uniform then q is negligible. So, if after the change H is queried on σ with non-negligibly probability then we can in the reduction make the guess that ρ is pseudorandom when H queried on σ and that it is uniform when H is not queried on σ. This guess can be made before we need τ for programming. In the final hybrid we then stop programming the random oracle, but instead let $\tau = H(\sigma)$. This is perfectly indistinguishable, so the advantage a'' does not change. Now we are back at the real execution, where everything is computed as in the protocol, and therefore $a'' = a'$. This concludes the proof. ■

B Proofs for seed

In this section we prove the following.

Theorem 2 *The scheme* seed *given in algorihm 3 and algorihm 4 is an unpredictable seed generation protocol as defined in section 4.* □

We first formalize unpredictability as a challenge game played against a PPT adversary.

Definition 4 (unpredictability) The adversary can win the following game with probability at most negligibly more than $2^{-\lambda}$. The adversary may initiate as many sessions sid as it wants. At some point it specifies (sid, g), where sid is a session where no honest party got input $(\textsc{seed}, \mathsf{sid})$. The adversary wins if it can execute to the point where some honest party outputs $(\textsc{done-seed}, \mathsf{sid}, c)$ with $c = g$. □

Termination. The only place where the protocol might deadlock is in line 56 if there are not $w_{\textsc{seed}}$ roles won by honest parties. This happens with at most negligible probability when committees are sampled as in Section 7. Namely, if a role is won by an honest party then eventually wVSS is run and will eventually give public output $\mathsf{PubOutVSSCommit}$. Note that the opening phase cannot deadlock by termination 2 of wVSS.

Agreement. Agreement follows from agreement on $\{\mathsf{sid}_1, \ldots, \mathsf{sid}_{w_{\textsc{seed}}}\}$ which is defined from public outputs on Ledger and agreement on r_{sid} for $\mathsf{sid} \in \{\mathsf{sid}_1, \ldots, \mathsf{sid}_{w_{\textsc{seed}}}\}$ which follows from Validity and Binding of wVSS.

Unpredictability. By choosing the parameters as in Section 7 we guarantee that there are a least $m_{\text{SEED}} - w_{\text{SEED}} + 1$ good dealers (honest parties who sampled a hiding committee) on the committee, meaning at least one among the first w_{SEED} sessions to give public output $\mathsf{PubOutVSSCommit}$. We will call this session sid^*. Since no honest party opens any commitment until the first w_{SEED} contributions outputted $\mathsf{PubOutVSSCommit}$ it follows from Hiding when good of wVSS that r_{sid^*} was unpredictable. For all honest sessions $\mathsf{sid}' \neq \mathsf{sid}^*$ we can imagine giving the $r_{\mathsf{sid}'}$ of these to the adversary. This only makes its job easier. For all corrupted sid' we have from Binding of wVSS that the adversary can compute $r_{\mathsf{sid}'}$ before any honest commitment is opened. Now guessing $\mathsf{seed} = \bigoplus_{\mathsf{sid}} r_{\mathsf{sid}}$ before any honest commitment is opened it PPT equivalent to guessing r_{sid^*}. So Hiding follows from Unpredictable of wVSS.

C Proofs for wHDCF

Definition 5 (Honest-Dealer β-Unpredictability for wHDCF) We formalize honest-dealer unpredictability as a challenge game played against the PPT adversary $\mathcal{A}$. The definition implicitly assumes the termination property, as it does not make sense to define unpredictability of a value which is not defined, and the agreement property, as it is trivial to guess one of the outcomes if two honest parties have different coins. The following definition demands that the dealing phase results in a "good" setup with a constant probability, and, if D is honest, such a good setup always leads to unpredictable coin flips.

Sometimes Good: For all sid there exists an event $\mathsf{Bad}_{\mathsf{sid}}$, which can be defined in PPT from the view of the adversary once the first honest party gives output $(\text{DONE-DEAL}, \mathsf{sid})$, i.e., the adversary knows if the event happened. It should hold that $\mathsf{Bad}_{\mathsf{sid}}$ happens with probability at most $1 - \beta$, independently for each sid and independently from D being honest.

Unpredictable when Good: Furthermore, the adversary can win the following game with probability at most negligibly better than $2^{-\lambda}$. The adversary may initiate as many sessions sid and coin-flips cid as it wants. At some point it specifies $(\mathsf{sid}, \mathsf{cid}, g)$, where sid is a session with an honest dealer, where $\mathsf{Bad}_{\mathsf{sid}}$ did not happen, where no honest party got input $(\text{FLIP}, \mathsf{cid})$ yet, and where $g \in \{0,1\}^\lambda$. The adversary wins if it can execute to the point where some honest party outputs $(\text{DONE-FLIP}, \mathsf{cid}, c)$ with $c = g$. □

We now prove the properties of the scheme. In the following, let f denote the actual number of corrupted parties in the coin-holding committee. Then, denote by GOOD-SETUP, NO-UNPRED, NO-LIVE the events that $f \in [0, \tau_{\text{COIN}}]$, $f \in (\tau_{\text{COIN}}, n_{\text{COIN}} - \tau_{\text{COIN}})$, and $f \in [n_{\text{COIN}} - \tau_{\text{COIN}}, n_{\text{COIN}}]$, respectively. As the names suggest, all protocol properties will be satisfied in the first case, while unpredictability may be violated in the second, and additionally liveness may be violated in the third. In Section 7 we choose concrete parameters for the committee election, such that $\Pr[\text{GOOD-SETUP}] = \beta$, for β a constant, and $\Pr[\text{NO-LIVE}]$ is negligible. Observe that $\mathsf{SampleCommittee}_p()$ is verifiable and unpredictable (see Section 2.1), hence D cannot affect the probability of these events.

PROOF (TERMINATION) (1) For the first part, assume D is honest. Since D does not wait for any parties in any step, it successfully broadcasts the coin setup, and, from the liveness property of Ledger, it will eventually be delivered on Ledger.

(2) Assume $(\text{DONE-DEAL}, \mathsf{sid})$ has been observed by all honest parties. Hence, honest parties can read the coin setup from Ledger and verify it using $\mathsf{VerifyCommittee}()$. By nature of committee

election and by the choice of n_{COIN} and τ_{COIN} there are at least $\tau_{\text{COIN}} + 1$ honest coin holders. We distinguish two cases. Either the dealer has created valid key shares and encryptions for *all* honest coin holders, or there is *at least one* honest coin holder, for whom the dealer has created an invalid key share. Observe that the two cases cover all possible executions. If we are in the first case, then every honest coin holder P_i will successfully decrypt e_i to get a valid sk_i. From *termination* property of CF, and since there are at least $\tau_{\text{COIN}} + 1$ honest coin holders, it follows that the coin shares of these parties are sufficient to reconstruct the coin value, hence eventually every honest party in the committee will output $(\text{FLIP}, \mathsf{sid}, \mathsf{cid}, \cdot)$. If we are in the second case, the honest party can always compute a valid complaint against the dealer. If e_i is incorrect, P_i can prove this by broadcasting a complaint COMPLAINTENCRYPTION, which contains a zk-proof W_i that e_i decrypts to (sk'_i, r'_i). A verifier can always verify the zk-proof and check that re-encrypting (sk'_i, r'_i) does not give e_i. If, on the other hand, e_i is a correct encryption of an invalid key share sk'_i, P_i can also prove this broadcasting sk'_i and r'_i in a COMPLAINTSHARE message. A verifier can now verify that (sk'_i, r'_i) indeed re-encrypts to e_i, but $\mathsf{CF.VerifyKeyShare}(\mathsf{vk}, i, \mathsf{sk}'_i)$ returns 0. Honest parties will eventually deliver the complaint and output $(\text{FLIP}, \mathsf{sid}, \mathsf{cid}, \perp)$. ■

PROOF (WEAK AGREEMENT) Let P and Q be parties that output $c_{\mathsf{P}} \neq \perp$ and $c_{\mathsf{Q}} \neq \perp$ as the value of the coin using sets of $t+1$ valid coin shares S_{P} and S_{Q} in $\mathsf{CF.Combine}()$, respectively. Then, since the shares in S_{P} and S_{Q} are valid, they all lie on the same polynomial, hence they define the same secret, and $c_{\mathsf{P}} = c_{\mathsf{Q}}$. Additionally, if D is correct, then no valid complaint can be computed, except with negligible probability, hence honest parties P and Q output $c_{\mathsf{P}} \neq \perp$ and $c_{\mathsf{Q}} \neq \perp$, and, from the *agreement* property of CF, we get $c_{\mathsf{P}} = c_{\mathsf{Q}}$. ■

PROOF (HONEST-DEALER β-UNPREDICTABILITY) Define $\beta = \Pr[\text{GOOD-SETUP}]$ and $\mathsf{Bad}_{\mathsf{sid}} = \neg\text{GOOD-SETUP}$, i.e., $\mathsf{Bad}_{\mathsf{sid}}$ is the event that the committee assigned to D contains more than τ_{COIN} corrupted parties. Given sid and a point p on Ledger, the committee returned by $\mathsf{SampleCommittee}_p()$ is deterministic, hence $\mathsf{Bad}_{\mathsf{sid}}$ happens with probability $1 - \beta$, which is constant and independent of sid, and can be defined from the view of the adversary once the coin setup appears on Ledger. The exact value of β depends on the choice of n_{COIN} and τ_{COIN}. The *Sometimes Good* property is satisfied. Now about the *Unpredictable when Good* property. When $\mathsf{Bad}_{\mathsf{sid}}$ does not happen, the number of actual corrupted coin holders is not more than τ_{COIN}. The *Unpredictability* property of the CF scheme holds in this case, and the *Unpredictable when Good* property of wHDCF can be reduced to the *Unpredictability* of CF. ■

D Proofs for wMDCF

Definition 6 (The *sometimes good* property) For all sid and cid there exists an event $\mathsf{Bad}_{\mathsf{sid},\mathsf{cid}}$, which is defined in PPT from the view of the adversary once the first honest party gives output $(\text{DONE-DEAL}, \mathsf{sid})$, i.e., the adversary knows if the event happened. Moreover, there exists a probability $\gamma > 0$, called the *good-setup probability*, which depends on the parameters m_{wMDCF} and w_{wMDCF} and on the hiding probability β of wHDCF, is constant and independent of sid and cid, and $\mathsf{Bad}_{\mathsf{sid},\mathsf{cid}}$ happens with probability $1 - \gamma$. □

We formalize agreement as a challenge game played against the PPT adversary $\mathcal{A}$. It assumes the termination property, so that the value of coin-flips actually become known.

Definition 7 (γ-Agreement) We require that the *sometimes good* property holds for $\mathsf{Bad}_{\mathsf{sid},\mathsf{cid}}$. Moreover, The adversary can win the following game with at most negligibly probability. The adversary may initiate as many sessions sid and coin flips cid as it wants. At some point it specifies $(\mathsf{sid}, \mathsf{cid}, \mathsf{P}, \mathsf{Q})$, where P has output $(\textsc{done-flip}, \mathsf{sid}, \mathsf{cid}, c_{\mathsf{P}})$, Q has output $(\textsc{done-flip}, \mathsf{sid}, \mathsf{cid}, c_{\mathsf{Q}})$, and sid and cid are such that $\mathsf{Bad}_{\mathsf{sid},\mathsf{cid}}$ did not happen. The adversary wins if $c_{\mathsf{P}} \neq c_{\mathsf{Q}}$. □

We formalize unpredictability as a challenge game played against the PPT adversary $\mathcal{A}$. As with wHDCF, the definition assumes the termination and agreement properties.

Definition 8 (γ-Unpredictability) We require that the *sometimes good* property holds for $\mathsf{Bad}_{\mathsf{sid},\mathsf{cid}}$. Moreover, the adversary can win the following game with probability at most negligibly better than $1/2$. The adversary may initiate as many sessions sid and coin-flips cid as it wants. At some point it specifies $(\mathsf{sid}, \mathsf{cid}, g)$, where $(\mathsf{sid}, \mathsf{cid})$ is such that $\mathsf{Bad}_{\mathsf{sid},\mathsf{cid}}$ did not happen, where no honest party got input $(\textsc{flip}, \mathsf{sid}, \mathsf{cid})$ yet, and where $g \in \{0, 1\}$. The adversary wins if it can execute to the point where some honest party outputs $(\textsc{done-flip}, \mathsf{sid}, \mathsf{cid}, c)$ with $c = g$. □

Proof (Termination) (1) The dealing protocol waits in two places, first for w_{wMDCF} instances of wHDCF, and then for an instance of $\textsc{seed}$. For the first one, the choice of parameters guarantees that, except with negligible probability, at least w_{wMDCF} instances of wHDCF will have an honest dealer, and each of these instances will terminate, according to the *termination* property of wHDCF. For the second, termination follows directly from the $\textsc{seed}$ protocol.

(2) As the Flip algorithm of wMDCF calls one of the wHDCF instances, the Flip termination of wMDCF directly follows from the Flip termination property of wHDCF. ■

Proof (Sometimes good) Let $\mathsf{Bad}_{\mathsf{sid},\mathsf{cid}}$ be the event that $\mathsf{Hash}(\mathsf{sid}, \mathsf{cid}, \mathsf{seed})$ points to a *bad coin setup*, i.e., to a wHDCF instance sid_j whose dealer is corrupted or whose coin-holding committee has more than τ_{COIN} corruptions (which happens with probability β). The value of seed, which is included as a parameter when hashing to obtain sid_j, is unpredictable by the adversary, by the unpredictability property of seed, and becomes known *after* the w_{wMDCF} coin setups have appeared on Ledger. Hence, the probability of $\mathsf{Bad}_{\mathsf{sid},\mathsf{cid}}$ happening is independent of cid, and only depends on the probability β of $\mathsf{Hash}(\mathsf{sid}, \mathsf{cid}, \mathsf{seed})$ hitting a bad committee. Hence, this event happens with probability $1 - \gamma$, which is constant and independent of sid, cid. ■

Proof (γ-Agreement) Let $(\mathsf{sid}, \mathsf{cid})$ such that $\mathsf{Bad}_{\mathsf{sid},\mathsf{cid}}$ did not happen, and hence the dealer of the sid_j CF instance, for $j = \mathsf{Hash}(\mathsf{sid}, \mathsf{cid}, \mathsf{seed})$, is honest. The property then follows from the *weak agreement* property of wHDCF. ■

Proof (γ-unpredictability) Let $(\mathsf{sid}, \mathsf{cid})$ such that $\mathsf{Bad}_{\mathsf{sid},\mathsf{cid}}$ did not happen. This means that the dealer of the sid_j CF instance, for $j = \mathsf{Hash}(\mathsf{sid}, \mathsf{cid}, \mathsf{seed})$, is honest. It also implies that the event $\mathsf{Bad}_{\mathsf{sid}}$, defined in the proof *honest dealer β-unpredictability* property of wHDCF as the event that the committee of the sid_j instance contains more than τ_{COIN} corruptions, did not happen. Hence, the property reduces to *Honest-dealer β-Unpredictability* of wHDCF. ■